IMPLEMENTACIÓN DE UN SISTEMA MULTIAGENTE PARA SOLUCIONAR EL PROBLEMA DEL AGENTE VIAJERO

Camilo Ernesto Charry Caicedo

Proyecto De Grado Presentado Como Requisito Parcial
Para Optar Al Título De Ingeniero Electrónico.

JAIME VELASCO MEDINA Ph.D.
Director de Proyecto

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
PROGRAMA ACADEMICO DE INGENIERÍA ELECTRÓNICA
SANTIAGO DE CALI
2012

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Santiago de Cali, Agosto 13 de 2012

CONTENIDO

Página

LISTA DE TABLAS.

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN.

Dentro del ámbito académico, se debe procurar permanecer siempre a la vanguardia con respecto a los avances tecnológicos y con ellos explorar las diferentes alternativas de uso de las mismas, para abrir así nuevas posibilidades de investigación y desarrollo a las generaciones futuras.

Algunas de estas tecnologías corresponden a ramas diferentes del conocimiento como la computación y la electrónica, encontrando así, de un lado los agentes inteligentes como una nueva y prometedora forma de solución a problemas complejos y de otro lado, se tiene los dispositivos de hardware reconfigurable FPGAs (*Field Programable Gate Array*) que cada día evolucionan e innovan, procurando dar soluciones efectivas a problemas reales y complejos que requieren de gran capacidad de procesamiento en forma paralela.

Dadas las características y ventajas de los desarrollos a nivel de hardware, en este trabajo se plantea la implementación de un sistema multiagente (SMA), para solucionar el problema del agente viajero y evaluar por medio de la comparación con una implementación software, la eficiencia del diseño.

1.1 MOTIVACIÓN.

Desde los inicios de la computación las diferentes técnicas o paradigmas de programación han ido evolucionando hacia el intento de conseguir soluciones a problemas complejos, tratando de emular la inteligencia humana de forma artificial (IA). Una de las últimas técnicas en este aspecto es el desarrollo del paradigma de sistemas multiagente, el cual demanda gran capacidad de procesamiento y mayormente el uso de paralelismo. Los sistemas multiagente, son implementados en software ejecutándose sobre plataformas con procesadores de propósito general que solo logran emular el paralelismo de forma poco eficiente, haciendo más lenta la ejecución de los programas debido al funcionamiento secuencial inherente a este tipo de procesadores. Es por este motivo que se presenta en este trabajo de grado la implementación de un sistema multiagente sobre hardware reconfigurable FPGAs, el cual ofrece las características de concurrencia adecuadas para explotar de forma eficiente las ventajas de los sistemas multiagente.

1.2 CONTRIBUCIÓN.

En este proyecto de grado se implementó tanto en hardware reconfigurable (FPGAs), como en software (JADE), un sistema multiagente para solucionar un problema típico y bien conocido como es **el problema del agente viajero (PAV)**, obteniéndose:

- La implementación de un sistema multiagente sobre la plataforma JADE para solucionar el problema del agente viajero valiéndose de un algoritmo genético paralelo maestro-esclavo, como motor de búsqueda.
- Un equivalente del paradigma de sistemas multiagente software aplicado a una implementación hardware.
- La implementación de un sistema multiagente sobre FPGAs usando la metodología BDI (*Belief Desires Intentions*).

- En la implementación hardware se logró una considerable disminución en el tiempo de obtención de una respuesta con respecto a la implementación software y mojorar el resultado hasta en un 2% con respecto a una solución optima del problema estudiado.

1.3 ORGANIZACIÓN.

Este trabajo de grado está organizado de la siguiente manera:

Capítulo 2- Se presentan las bases teóricas y herramientas para el trabajo con sistemas multiagente software.

Capítulo 3- Se describen algunos algoritmos para la solución del problema del agente viajero y las similitudes y facilidades de implementación en un sistema multiagente.

Capítulo 4- Se presenta el sistema multiagente software para diferentes casos del PAV y se compara las soluciones con otros métodos conocidos.

Capítulo 5- Se presenta el desarrollo del sistema multiagente hardware, su aplicación a diferentes casos de PAV y la comparación de desempeño con la implementación software.

Capítulo 6- Se presentan las conclusiones y el trabajo futuro.

CAPÍTULO 2

TEORÍA DE AGENTES Y SISTEMAS MULTIAGENTE.

La formación teórica de los agentes nace a mediados de los años 70's[1], cuando los investigadores se plantean la posibilidad de resolver problemas complejos de forma inteligente, mediante la cooperación entre múltiples sistemas**,** dando lugar a la aparición de una nueva rama de trabajo dentro de la inteligencia artificial conocida como inteligencia artificial distribuida (IAD). El resultado de todas estas investigaciones es una arquitectura de sistemas cooperantes entre fuentes de conocimiento mediante un modelo de pizarra o tablero para realizar los procesos de comunicación entre las diferentes entidades que pueden trabajar en paralelo o de forma independiente y descentralizada, pero que requieren de un control para que los diferentes procesos se hagan en el tiempo y en la secuencia correcta para que las soluciones parciales dadas por los sistemas cooperantes[1] puedan dar una solución total válida. En la década de los 80 se consolida el término de agente para identificar las entidades que cooperan para la solución de un problema, dando inicio así a un nuevo paradigma de programación, el *Agent Based Computing*[2] *(ABC),* y a su vez nacen nuevos cuestionamientos de lo que debería ser o no un agente**,** cuáles son sus capacidades y aéreas de aplicación, cómo se clasifican y qué arquitecturas deberían tener para ser considerados como agentes.

[1] Los sistemas cooperates se diferencian de otros sistemas más antigos básicamente en la capacidad de comunicación en interación con otros sistemas.
[2] En este paradigma se emplea el método de tablero para el paso de mensajes entre las entidades cooperantes.

En este sentido la tecnología de agentes sigue siendo aún muy ambigua debido a que los teóricos no se han puesto totalmente de acuerdo en cuanto a definiciones y propiedades, sin embargo hay tendencias que son más ampliamente aceptadas que otras y esas son la que se asumen en este trabajo[3].

2.1 CONCEPTO DE AGENTE.

Un agente inteligente, es una entidad software o hardware capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional, es decir, de manera correcta con tendencia a maximizar un resultado esperado.

Teniendo en cuenta que el concepto de racionalidad es más general para describir el comportamiento de los agentes inteligentes y por ello más adecuado que el de inteligencia, la cual sugiere entendimiento, los agentes inteligentes se describen esquemáticamente como un sistema funcional abstracto, razón por la que a veces son llamados Agentes Inteligentes Abstractos (AIA) para distinguirlos como sistemas informáticos de otros tipos de agentes[4]. En Ciencias de la Computación el término agente inteligente puede ser usado para referirse a un agente de software que tiene algo de inteligencia. Por ejemplo, programas autónomos utilizados para asistencia de un operador o de minería de datos son también llamados **agentes inteligentes.**

En general algunas de las propiedades[2][3] o al menos las más aceptadas que deben tener los programas para que sean llamados agentes (**figura 1**) son:

- **Autonomía:** puede ser tomada en un sentido amplio como la imprevisibilidad del comportamiento del agente o a la capacidad del

[3] En cada trabajo de investigación el autor propone su propia definición.
[4] Agentes bilógicos, agentes robots, agentes policiales etc.

agente para elegir las acciones adecuadas para el cumplimiento de sus objetivos sin intervención externa[5].

- **Sociabilidad:** los agentes pueden comunicarse y tener interacción con otros agentes haciendo uso de un lenguaje determinado[6].
- **Reactividad:** denota la capacidad de percibir cambios en el entorno y reaccionar a estos.
- **Proactividad:** los agentes pueden actuar en busca de un objetivo sin necesidad de recibir ningún estímulo.
- **Benevolencia:** los agentes están dispuestos a ayudar a otros agentes y prestar sus servicios si ésto no está en contra de su objetivo.
- **Veracidad:** parte del hecho que un agente siempre comunica información veraz o en su defecto, no proporciona información falsa con conocimiento de hecho[7].

Figura 1 Propiedades más aceptadas de un agente.

[5] No es necesario que una persona esté impartiendo directivas con el ánimo de guiar al agente por un camino, el agente decide que camino es el correcto según sus directivas de diseño.
[6] La sociabilidad no solo se refleja en la interaccion con otros agentes sino también con los humanos.
[7] La veracidad esta relacionada con lo que el agente considera verdadero con respecto del limitado conocimiento que posee de su entorno.

Se debe tener en cuenta que no hay una definición completamente aceptada de lo que debe ser un agente, puesto que las propiedades pueden variar, sin embargo éstas son las más relevantes y están presentes en la mayoría de las definiciones aceptadas en la literatura. De igual forma son muchos los tipos de agentes que se pueden crear a partir de las definiciones de estas propiedades, del tipo de aplicación o del medio al cual vaya a ser aplicado.

2.2 SISTEMAS MULTIAGENTE.

Es un grupo de agentes que conviven en un entorno al que perciben y conocen de forma total o parcial y al que pueden modificar con base en ese conocimiento para producir resultados que obedezcan a las condiciones de su diseño maximizando sus resultados. Un SMA está compuesto por más de un agente, los cuales pueden ser homogéneos o heterogéneos y trabajar en conjunto o en competencia, para alcanzar un objetivo común maximizando los resultados.

Un sistema multiagente funciona de forma muy similar como podría hacerlo una empresa, un grupo de individuos que trabajan de forma coordinada en la realización de un objetivo común, tratando de optimizar el uso de los limitados recursos a su disposición. En este sentido, diseñar un SMA guarda un cierto paralelismo con el diseño de la estructura de una empresa, con la diferencia que en lugar de individuos se tendrán agentes, programas con comportamiento similar al de los individuos y en lugar de edificios y/o departamentos se tendrán ordenadores conectados en red**.**

2.3 METODOLOGÍAS DE DESARROLLO DE SISTEMAS MULTIAGENTE.

Básicamente una metodología es una secuencia de pasos a seguir con el fin de construir sistemas multiagente de forma organizada y repetible.
En la actualidad existen diversas metodologías para el diseño de sistemas multiagente, aunque no muchas de ellas vienen acompañadas de herramientas o

arquitecturas que soporten el diseño, lo cual hace más difícil su implementación y en algunos casos los proyectos han sido de alguna forma abandonados o no existe documentación suficiente. Debido a que el exponer todas las metodologías existentes está por fuera del propósito de este trabajo, se hace entonces una aproximación a las metodologías más relevantes[8] para el proyecto y que se emplearon durante su desarrollo.

2.3.1 Metodología BDI.

Esta metodología no es una extensión ni deriva de otra tal como una orientada a objetos, está hecha con base en un modelo cognitivo[9] del ser humano, donde el agente está dotado de deseos, creencias e intenciones, percibe su entorno a través de sensores y delibera para interactuar con éste con base en lo que percibe, en lo que cree y en lo que quiere[4].

Las creencias representan el modelo del mundo, los deseos son los objetivos a los que el agente debe llegar y las intenciones son el camino o las acciones que puede tomar para alcanzar lo que desea, en este caso alcanzar los objetivos de diseño. La visión general del sistema está dividida en dos partes, una vista interna y una vista externa del sistema. La vista interna se enfoca en el funcionamiento individual de los agentes, en la cual éste está definido completamente por lo que puede percibir, lo que puede hacer, los objetivos que puede perseguir y los planes que puede llevar acabo, todo esto encapsulado en tres modelos:

- Modelo de creencias, en el cual se tiene información concreta acerca del entorno y las acciones que el agente puede ejercer sobre éste.

[8] La metodología escogida para el desarrollo de este trabajo en cuanto a la implementación hardware es la BDI, la cual se adapta por medio de un paralelo comparativo hardware-software, ver ANEXO A.

[9] En general los sistemas de agentes son llevados a la abstracción haciendo un simil con características sicológicas del ser humano, y no es raro atribuirle a los agentes propiedades como, conocimiento, creencias, intenciones, deberes, etc.

- Modelo de deseos, en el que están los objetivos a los que el agente quiere llegar y los eventos a los que puede responder.
- Modelo de intenciones, en el cual se especifican los planes que el agente puede seguir para la consecución de los objetivos.

Estos tres modelos por si solos no son suficientes para dar soporte a un sistema multiagente, por lo cual se requiere de la vista externa, que capture las interacciones, responsabilidades y servicios que proveen los agentes, capturando éstos en dos modelos modelos adicionales (**Figura 2**):

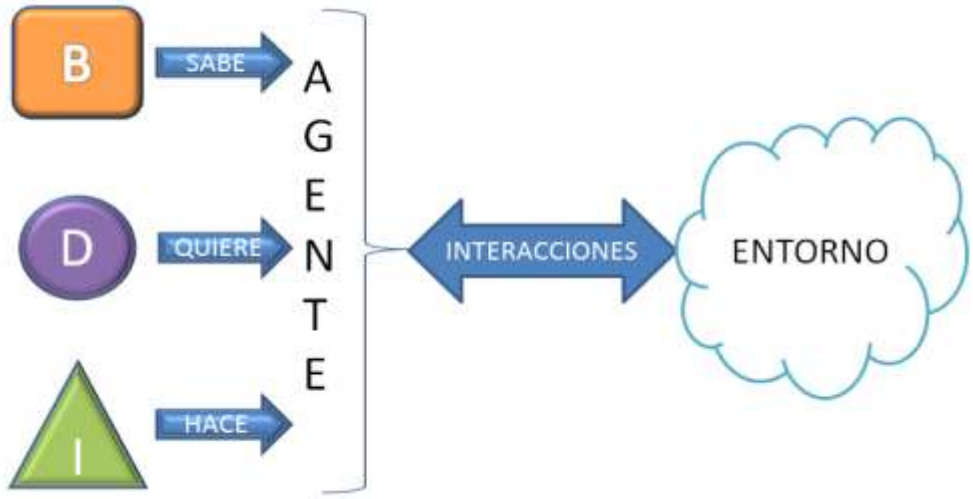
- Modelo del agente, donde se describe la jerarquía de clases, el número de agentes y su multiplicidad así como el tiempo en el que existirán.
- Modelo de interacción, en el cual se definen las responsabilidades, la forma en que los agentes se relacionan unos con otros y la sintaxis que usan para la comunicación.

Esta metodología proporciona grandes beneficios en cuanto a robustez y estabilidad de las aplicaciones, además, se pueden añadir o quitar comportamientos o planes para realizar cambios significativos en el agente sin causar mayor alteración a los ya existentes. Muchas de las metodologías están hechas con base en modelos anteriores,[10] como metodologías orientadas a objetos que en cuestiones de rigor de diseño teniendo en cuenta el ciclo de vida del software están mucho más detalladas y con un buen soporte debido a sus predecesoras, pero hasta ahora la única que engloba todo lo que represente un agente y por consiguiente un sistema compuesto por múltiples agentes es la BDI, que rompe con los paradigmas tradicionales de diseño, además el nivel de

[10] Debido a que los sistemas de agentes están en desarrollo, se han adoptado metodologías que tienen gran trayectoria y han sido ampliamente estudiadas ba jo otros paradigmas como son los orientados objetos.

abstracción lo hace de fácil implementación sobre hardware[11], ya que se ocupa del diseño del agente como unidad independiente y no como un todo que involucra el problema completo, también las interacciones entre agentes dado por la vista externa del sistema, permite la aplicación de diferentes tipos de técnicas para conformar el sistema de forma independiente a la del diseño de los agentes.

Figura 2 Arquitectura BDI.



2.3.2 Metodología *VOWEL ENGINEERING.*

Esta metodología está hecha con base en 4 unidades básicas que son denominadas por vocales de la forma: A, de agentes, E, de entorno, I, de interacción, y O, de organización. Teniendo en cuenta esta división se puede aplicar técnicas de modelado independientes para cada subgrupo, teniendo de esta forma la posibilidad de representarlos de manera simple o por sistemas más complejos. El principal objetivo de esta metodología es el de desarrollar componentes de forma individual en cada uno de los subgrupos de forma tal que

[11] Es mas fácil aplicar este tipo de abstracciones a un sistema físisco, como un circuito, que sobre una entidad software que ya esta predeterminada para ciertas metodologías, incrementando de esta manera la complejidad de la misma.

los diseñadores tengan la posibilidad de escoger un componente que ya haya sido empleado y construir uno nuevo a partir de estos[5].

Al desarrollar un sistema siguiendo esta metodología se debe realizar cada etapa en un orden establecido por la importancia de éste dentro del mismo, i.e. si lo que más importa es la parte de comunicación entonces se debe empezar por la I, o si lo más importante es la organización entonces se empieza por la O, al teminar, el objetivo es obtener una serie de librerías de componentes que solucionen cada uno de los aspectos y así el diseñador pueda seleccionar un modelo de agente, uno de entorno, uno de comunicación y uno de organización que en conjunto solucionen el problema planteado, además que estos componentes pueden ser reutilizados y reciclados de forma individual.

En publicaciones recientes se propone el uso de la plataforma Volcano la cual utiliza el ensamblaje de componentes por medio de UniCon[12], donde cada componente corresponde a una de las cuatro categorías que conforman la metodología. La importancia de esta metodología viene dada por la gran variedad de aplicaciones en que puede ser empleada, según el trabajo de investigación del grupo MAGMA[13].

A pesar de ser una de las primeras metodologías para el modelado de sistemas multiagente, está incompleta ya que no tiene ninguna herramienta de soporte estable y no existe una descripción concreta de cómo afrontar cada uno de los aspectos. Como ventaja de esta metodología está la forma que ve al sistema como una composición de arquitecturas, lo cual facilita la reutilización código.

[12] Lenguaje de composición de componentes.
[13] Grupo de investigación dedicado al modelado y simulación de sistemas sociales complejos a través de la implementación de sistemas multiagente.

2.3.3 Metodología MaSE[14].

Esta es una metodología bastante completa, con base en el ciclo de vida del software, que incluye una herramienta de desarrollo propio la cual se divide en dos fases: el análisis y el diseño[6][5].

A su vez la fase de análisis se subdivide en:

- Captura de objetivos
- Casos de uso
- Refinamiento de roles

La fase de diseño se subdivide en:

- Clases de agente
- Conversaciones
- Ensamblaje de agentes
- Diseño del sistema

Los agentes no se consideran como entes autónomos sino más bien como procesos que se intercomunican para conseguir un objetivo global y podrían tener o no inteligencia, se puede decir que los agentes son objetos especializados y el sistema se construye sobre tecnología O.O. y la aplicación se orienta a la especificación y diseño de sistemas multiagente.

La herramienta con la que cuenta la metodología MaSE, el AgentTool[15], soporta todas las etapas de diseño de forma gráfica, además de permitir generar código de forma automática teniendo como salida un archivo de texto haciéndolo independiente de la plataforma de trabajo. Aunque parezca sencillo, hay dependencias entre los diferentes bloques y realmente no es nada simple el saber

[14] *Multi Agent systems Software Engineering.*
[15] Esta herramienta permite definir un agente o un sistema de agentes por medio de la descripción gráfica del sistema sobre JAVA, creando la estructura principal de forma automática y dejando el resto al programador.

cuáles máquinas de estados definen la ejecución de una tarea contextualizada dentro de un conjunto de estas.

2.3.4 MAS-CommonKADS.

Proviene de una metodología enfocada en la construcción de sistemas expertos, la cual aprovecha la experiencia de ésta en cuanto a las ideas orientadas a objetos y se convierte en la primera en proponer la integración de los sistemas multiagente con un modelo de desarrollo software estándar (espiral dirigido por riesgos)[6][7][5].

Se plantea dividiendo el sistema en siete modelos diferentes: agente, tareas, coordinación, organización, experiencia, comunicación y diseño, este último engloba el resultado de todos los demás, además de que el modelo de agente está relacionado con todos de forma directa y está presente en todo el desarrollo convirtiéndose en el más importante. Cada modelo parte de una descripción gráfica complementada por una descripción en lenguaje natural. Es una metodología bastante meticulosa en cuanto a la descripción de cada uno de sus componentes teniendo en cuenta sus dependencias, en la que se ha hecho un gran esfuerzo por cubrir todo el ciclo de vida del software, teniendo como principal desventaja, la falta de una herramienta de desarrollo ya que no es fácil hacerla de forma automática.

2.3.5 Metodología GAIA.

En el modelo GAIA se tiene una distinción entre la parte abstracta y que no se refleja directamente en la implementación, separando la parte de análisis de la de diseño. En la fase de análisis se tiene como objetivo desarrollar y comprender el sistema, sin entrar en detalle de la implementación y se divide en el modelo roles y el modelo de interacciones[2].

Los roles son una descripción abstracta de la función que se espera que cumpla una determinada entidad, están compuestos por permisos que definen los recursos que están a su disposición y responsabilidades que representan todas las tareas para las que fue creado, es decir los objetivos que éste debe cumplir.

En el modelo de interacciones se tratan las dependencias entre los roles las cuales son un punto crítico dentro del sistema, tratándolas como un conjunto de definiciones y protocolos para la comunicación entre entidades y roles.

La etapa de diseño tiene como objetivo la de llevar el resultado del análisis a un modelo con detalle suficiente como para aplicar técnicas de diseño orientado a objetos para una posterior implementación, para ello se divide en tres modelos diferentes: agentes, servicios y conocidos.

En el modelo de agentes se documenta los diferentes tipos de agentes que harán parte del sistema, considerando al agente como un conjunto de roles empaquetados por el diseñador donde entre ellos se guarda algún tipo de relación.

En el modelo de servicios se identifican y se caracterizan las principales propiedades de estos y de cómo se asocian a un determinado conjunto de roles, entendiéndose por servicio, la utilidad del agente, siendo necesario la documentación de las propiedades de cada uno teniendo en cuenta las entradas, salidas y restricciones, las primeras derivando del modelo de interacción y las segundas del modelo de roles.

El modelo de conocidos define la comunicación existente entre los diferentes agentes que componen el sistema, indicando la vía de comunicación sin tener en cuenta el mensaje en sí, donde el objetivo principal es la identificación de cuellos de botella en las comunicaciones y así evitar problemas en tiempo de ejecución.

2.4 PLATAFORMAS DE DESARROLLO PARA SISTEMAS DE AGENTES.

Los lenguajes de agentes han sido hasta el momento desplazados por el uso de plataformas de desarrollo[16] o *frameworks,* cuya base de programación es algún lenguaje orientado a objetos debido a que este paradigma muestra soporte para muchas de las características de los sistemas de agentes, estas plataformas son diseñadas para dar soporte a metodologías particulares.

2.4.1 Plataforma JADE[17].

Muy seguramente es la plataforma para el desarrollo de sistemas multiagente más utilizada en el ámbito académico; provee una serie de clases que soportan la creación, ejecución, terminación y depurado de sistemas multiagente; da soporte a la comunicación entre éstos y proporciona un sistema de páginas amarillas donde agentes pueden buscar a otros agentes y los servicios que éstos proporcionan cuando se encuentren registrados en el sistema[8][9][10].
Pese a que para la construcción de los agentes, el desarrollador deba adaptarse a un estándar (**FIPA**), en el cual la forma de comunicación ya está definida y el resto de la lógica queda en manos del programador.

2.4.2 JADEX.

Jadex es un framework construido sobre jade para funcionar como maquinaria de razonamiento en agentes con metodología BDI. Proporciona herramientas de creación, comunicación, y administración de agentes así como de depuración y visualización gráfica de las comunicaciones y estados internos del sistema.

[16] También llamados midleware, los cuales están hechos con base en un paradigma O.O. pero que dan soporte a los sitemas de agentes.
[17] *Java Agent DEvelopment framework.*siendo esta la plataforma seleccionada para ra el la implementación software. Ver ANEXO A.

Actualmente se encuentra en desarrollo y ha sido liberada la versión 2.0 aunque la documentación aún no está disponible. Es la implementación oficial del estándar FIPA, el cual es uno de los estándares para sistemas multiagente más conocido y aceptado como tal, recordemos que hasta el momento no hay una definición concreta en cuanto a agente se refiere y los estándares tratan de solucionar esta situación[11].

2.4.3 ZEUS.

La plataforma ZEUS provee un kit de herramientas gráficas para construir sistemas multiagente distribuidos, en un ambiente para aplicaciones rápidas con una maquinaria de reglas, planificación y visualización. El entorno puede generar código a partir de la definición del sistema de forma gráfica. No se encuentra la documentación y links de descarga páginas desactualizadas.

2.4.4 JAFMAS[18].

Aparte de proveer el *framework* para el desarrollo de agentes también trae consigo su propia metodología de desarrollo a la que los programadores pueden someterse o no, esta metodología es genérica y está basada en actos del habla y posee un conjunto de clases para el soporte de los agentes en java. La finalidad de esta herramienta es la de proveer a los desarrolladores la ayuda para estructurar ideas y poder llevarlas a aplicaciones de sistemas de agentes concretas. Al parecer el desarrollo se encuentra abandonado pues las páginas no se actualizan desde 1997 y no se encuentran links de descarga para la aplicación ni documentación[19].

[18] *Java Framework for Multiaget Systems.*
[19] Muchos de los proyectos de investigación acerca de los sistemas de agentes y herramientas para su desarrollo han sido abandonados.

2.4.5 MadKit[20].

Es una plataforma de código abierto modular y escalable escrita en java y construida sobre un modelo organizacional especial para la creación de sociedades artificiales, no se encuentra asociado a ninguna metodología así que queda al criterio del desarrollador, puede ser empleada en algoritmos bio inspirados como colonias de hormigas o abejas o en cualquiera que involucre una organización social.

[20] *MultiAgent Development Kit.*

CAPÍTULO 3

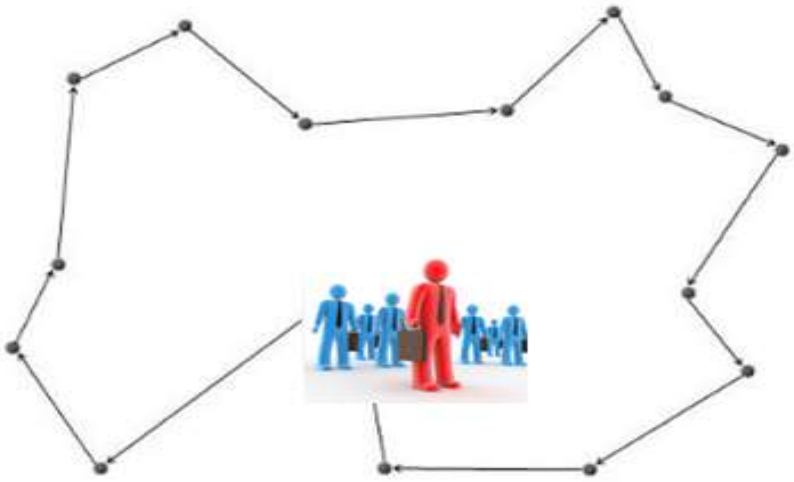
SOLUCIONANDO EL PROBLEMA DEL AGENTE VIAJERO.

El PAV, consiste en que un comerciante debe visitar un número *N* de ciudades una sola vez[21](**Figura 3**), partiendo de una ciudad y regresando a la misma minimizando el costo de recorrido. El PAV es probablemente el problema de optimización combinatoria más estudiado debido a su simple formulación y a la notoria dificultad de encontrar una solución eficiente. Para efectos de claridad se asume que el problema a solucionar es el PAV simétrico[22] en el que la distancia de A a B es igual que la distancia de B a A, este problema tiene solución pero no siempre en un tiempo razonable debido a la cantidad de operaciones que se deben hacer para encontrar la ruta más corta, las cuales están dadas por el factorial del número de ciudades involucradas en el problema concreto, es decir, la dificultad radica en el número de ciudades que el comerciante tenga que visitar, de este modo, entre más ciudades, más difícil es encontrar la solución en el espacio de búsqueda o al menos una solución aproximada que se precie de ser aceptable. Es por eso que los algoritmos exactos a pesar que aseguran la solución más óptima, dejan de ser prácticos después de un número relativamente pequeño de ciudades ya que con solo 10 de ellas se deben hacer 3628800 permutaciones.

[21] Este problema fue definido por W.R. Hamilton en 1930, donde se empieza su estudio matemático.
[22] Hay diferentes variaciones del problema en los que se incluyen problemas asimétricos y/o con multiples restricciones.

La naturaleza del problema hace que solo se pueda tener una solución aproximada en un tiempo razonable, por ello el esfuerzo de conseguir cada vez algoritmos más eficientes ha hecho que la cantidad de éstos sea abrumadora y que en este trabajo solo se mencionen unos pocos cuya relevancia es considerable para el desarrollo del proyecto, en este caso, los algoritmos heurísticos[23], los cuales son métodos que brindan soluciones aproximadas y en el mejor de los casos la mejor solución sin comprobación de que así sea.

Figura 3 Problema del agente viajero.



3.1 ALGORITMO VECINO MÁS PRÓXIMO.

Se trata de pararse en un nodo y buscar a partir de ahí cúal es el que está más cerca, cambiar de posición hacia éste y repetir la operación hasta que ya no queden nodos por visitar. Realmente es una tarea bastante secuencial y rápida que da una aproximación a la solución en un tiempo razonable.

[23] Algoritmos basados en reglas que descartan por hecho el objetivo primario de obtener la mejor solución a un problema.

3.2 GREEDY.

El algoritmo de *Greedy* encuentra un óptimo local en cada paso, con la esperanza de encontrar el óptimo global, sin embargo no siempre se obtiene el resultado esperado, pero se encuentra una solución óptima local próxima a la global en un tiempo razonable.

3.3 BORUVKA Y QUCIK BORUVKA.

Este algoritmo encuentra el mínimo número de aristas que conecta un número de vértices dentro de un grafo cuyos arcos poseen valores diferentes, éste también es conocido como algoritmo de *sollin.* El Algoritmo Quick Boruvka intercambia precisión por velocidad prescindiendo del ordenamiento de nodos cercanos.

3.4 ALGORITMO GENÉTICO.

Los algoritmos genéticos[24] son técnicas de optimización basadas en la forma en que las especies en la naturaleza han evolucionado, realizando una selección de los mejores especímenes para adaptarse a su entorno y así poder sobrevivir. Teniendo en cuenta operaciones básicas como son la reproducción, la herencia genética y la mutación se puede cubrir en tiempos razonables grandes espacios de búsqueda en cada iteración, en estas bases teóricas los algoritmos genéticos se han convertido en una alternativa para solucionar problemas complejos donde la cantidad de operaciones y el tiempo que toma llevarlas a cabo se vuelven inmanejables en la medida que el espacio de búsqueda se amplía.

Los algoritmos genéticos se valen de 3 operaciones básicas que se presentan en la naturaleza a la hora de evolucionar una especie a través de las diferentes generaciones, el cruce, la mutación y la selección, con las cuales las características de los padres se transmiten a su descendencia, reproduciendo las

[24] Planteados por primera vez por J.H. Holland en los años 70´s

mejores creando con el tiempo individuos mejor adaptados a su entorno, eliminando de la información genética las características menos deseables para una mejor adaptación.

3.4.1 Población.

Una población está compuesta por un número determinado de individuos, cada individuo representa dentro del algoritmo una posible solución al problema que se está estudiando, en este caso en particular cada individuo representa una posible ruta que minimiza la distancia del recorrido total del comerciante.

3.4.2 Operación de cruce.

El cruce es la operación genética más importante, ya que es la que recorre el espacio de búsqueda creando dentro de la población nuevos individuos a partir de individuos ya existentes, que poseen una probabilidad de cruce *Pc*, la cual por lo general suele ser bastante alta entre el 70% y el 90% . El cruce debe cumplir con las restricciones impuestas por el problema estudiado, en el caso del TSP las rutas deben recorrer el total de ciudades, empezar y retornar a la ciudad de inicio pasando solo una vez por cada una de ellas. En la literatura se encuentran diferentes técnicas para realizar el cruce entre individuos, entre los que se encuentran el cruce de un punto, cruce de dos puntos y cruce uniforme, en la **tabla 1** se muestra un ejemplo de cruce de un punto.

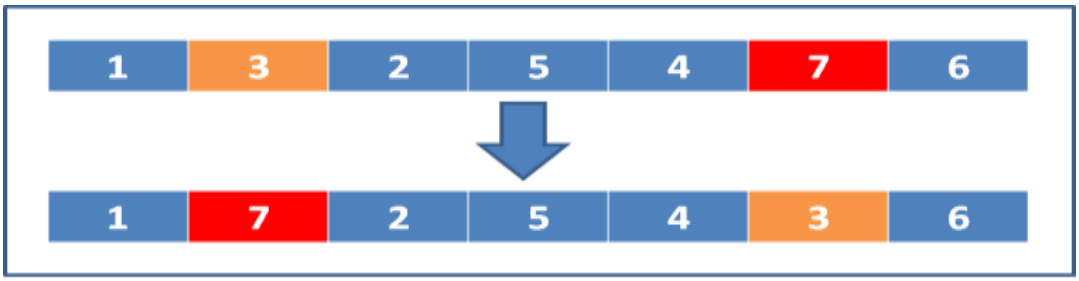
Tabla 1 Operación de cruce.

P0	2	3	6	5	4	1	7			
P1	4	7	2	5	1	3	6			
OX	4	7	2	5	1	3	6	6	5	4
H1	7	2	1	3	6	5	4			

3.4.3 Operación de mutación.

La operación de mutación suele tener muy poca ocurrencia dentro del algoritmo, sin embargo ésta va tomando importancia a medida que se avanza dentro de la ejecución del mismo debido a que la población tiende a homogenizarse y a caer en posibles mínimos locales, esta operación se encarga que el espacio de búsqueda sea explorado por completo, generalmente la mutación cambia uno de los cromosomas del gen alterando el orden de la solución con una probabilidad de ocurrencia *Pm* muy baja que para codificaciones binarias se encuentra entre el 0.5% y 2% En la **tabla 2** se muestra la mutación aplicada de sobre un gen.

Tabla 2 Operación de mutación.



3.4.4 Operación de selección.

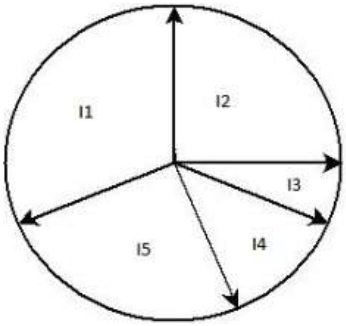
La selección es el proceso en el que se eligen los padres de los individuos que van a conformar la siguiente generación, para esto se tiene en cuenta qué tan bien adaptado se encuentre cada individuo dentro de la población donde la probabilidad de ser escogido para reproducirse suele estar asociada a esta adaptación, sin embargo el método de selección debe dar prioridad a los individuos mejor adaptados sin dejar completamente de lado a los menos favorecidos.

Al igual que las otras operaciones, la selección tiene diversos métodos de implementación siendo el método Montecarlo o de ruleta el más empleado, el cual consiste en asignar un valor ponderado equivalente al valor de adaptación de cada

individuo asignado a una ruleta, donde los individuos con mejor adaptación tienen más probabilidades de ser escogidos que los que no la tienen. Otro método de selección bastante conocido y que supera al método de la ruleta en ciertos aspectos[25] es el de selección por torneo, en el cual se escoge un número de individuos de forma aleatoria dentro de la población y se someten sus funciones de ajuste a un torneo, donde el ganador será el que tenga la mejor adaptación, con este método se puede llegar a dos cuestiones, cuáles son los mejores individuos y así poder pasar sus genes a la siguiente generación, y cuáles son los peores, aprovechando esta característica para reemplazarlos por los nuevos individuos que surgen de las operaciones de cruce y mutación entre los ganadores del torneo, aunque el número de seleccionados puede ser cualquiera, es muy corriente usar solo dos en cada selección.

En la figura 4 se muestra un ejemplo donde cada cuadrante de la "torta" representa la probabilidad de escogencia de un individuo dentro de la población donde los individuos mejor dotados son el I1, I2 e I5 los cuales tienen mayores posibilidades de reproducción para la siguiente generación, sin embargo lo demás individuos tienen probabilidad no nula y también pueden en cierto momento ser escogidos al lanzar la ruleta.

Figura 4 Representación del método de Montecarlo o de la ruleta para selección de individuos.



[25] Es posible evitar que un individuo con pocas aptitudes sea seleccionado para reproducirse más de una vez, por el contrario el método de la ruleta no tienen esta restricción haciéndola menos fiable.

3.4.5 Fitnes.

Es la función que da el valor de cada individuo y con ella se puede evidenciar que tan bueno es. Se encuentra de tipo puro, estandarizado y normalizado y depende directamente del problema que se esté tratando.
El fitnes de tipo estandarizado se usa en casos donde se desee maximizar o minimizar una función o problema dado.

3.4.6 Inicialización de la población.

Es un paso muy importante dentro del algoritmo en el que se genera la primera generación de individuos a partir de los cuales se comienza la búsqueda, se suele hacer de forma aleatoria aunque también se pueden emplear algoritmos heurísticos con el fin de reducir el espacio de búsqueda, buscando una convergencia prematura de la población y reducir el número de pobladores e iteraciones requeridas para encontrar una solución que sea aceptable.

3.5 ALGORITMO GENÉTICO PARALELO MAESTRO ESCLAVO.

Este algoritmo cuenta con dos tipos de entidades teniendo un maestro y uno o varios esclavos, cada uno tienen asociada unas tareas específicas, donde el maestro se encarga de inicializar la población, pasarla a los esclavos y esperar que éstos efectúen una generación completa recibiendo como parámetro una nueva generación por cada uno de los esclavos, paso seguido evalúa a cada individuo y selecciona el mejor, devolviéndolo como argumento una vez más a sus esclavos completando así una generación, los esclavos llevan a cabo las operaciones de selección, cruce y mutación a partir de la población inicial y cada uno puede tener probabilidades diferentes cubriendo así entre todos una amplia zona de búsqueda en cada nueva generación[12][13][14].

CAPÍTULO 4

IMPLEMENTACIÓN SOFTWARE: SISTEMA MULTIAGENTE PARA SOLUCIONAR EL PROBLEMA DEL AGENTE VIAJERO.

Para el desarrollo de este proyecto se escogió el algoritmo genético paralelo maestro esclavo, dada su similitud a un sistema multiagente, debido a que éste puede interpretarse como varias entidades software que se dedican a una tarea específica que en forma cooperante solucionan un problema, teniendo entre sí una comunicación definida e independencia en su ejecución[15].

4.1 HERRAMIENTAS DE DISEÑO.

Para la realización de este trabajo se emplearon las herramientas software para la edición, depuración y ejecucion de la solución propuesta:

- Java SDK 6.0
- Eclipse 3.6.2
- EJADE 0.9
- JADE 4.0
- CONCORD 1.1

En la implementación del sistema se usa el framework JADE 4.0 sobre JAVA el cual posee una amplia documentación y se encuentra información en línea fácilmente de forma gratuita, además de estar aun en desarrollo. Este framework

provee al usuario una interfaz para la creación, depuración y ejecución de agentes, así como de clases de comunicación usando el estándar FIPA (*Foundation for Intelligent Physical Agent*) dejándole al programador la tarea de programar el razonamiento interno.

Para el trabajo con la plataforma JADE se utilizó el editor ***Eclipse*** con el complemento EJADE[26] para edición y depuración de agentes inteligentes.

Para la prueba de comparación se empleó el software ***Concord*** [referencia] de libre distribución que implementa diferentes algoritmos de solución para el problema del agente viajero.

4.2 IMPLEMENTACIÓN.

El sistema multiagente se diseñó teniendo en cuenta las cualidades de la plataforma JADE y la similitud entre el algoritmo genético paralelo maestro esclavo, tomando en el sistema dos tipos de agente, un agente maestro y uno o varios agentes esclavo.
Esta implementación se modela cada agente como una máquina de estados finita (FSM), que determina el comportamiento y estado mental de cada agente, facilitando de esta manera su implementación en hardware (**figuras 5**, **6**).

4.2.1 Agente maestro.

Este agente recibe como parámetro el problema a solucionar siendo este un problema de tipo euclidiano en dos dimensiones y simétrico. El agente extrae la información requerida para solucionar el problema, que viene dada por el número de ciudades a recorrer y la posición relativa de dichas ciudades expresadas como un par ordenado, (estado S_0). Teniendo en cuenta que lo que se debe minimizar es la distancia recorrida. El siguiente paso es el de crear una primera generación

[26] Ejade es un complemento de eclipse para la edición y depuración de sistemas de agentes JADE.

de individuos, para esto se emplea el algoritmo heurístico *Nearest Neighbor*, que ante condiciones diferentes proporciona salidas diferentes, situación deseable para un algoritmo genético ya que, a pesar de limitar el espacio de búsqueda no hace converger la población total tras ser aplicado a la misma (estado S_1). Una vez obtenida la población inicial el agente consulta en el directorio de agentes[27] DF acerca de la didponibilidad de agentes para prestar el servicio requerido, para este caso, el de solucionar el TSP. El agente maestro recibe y guarda los nombres y direcciones suministrado por el DF de los agentes disponibles y así poder comunicarse con ellos, enviando la información recaudada en la primera instancia (estado S_2), en siguida la información es enviada (estado S_3), usando mensajes ACL (*Agent Communication Language*) y se espera de retorno las mejores soluciones de cada uno de los agentes (estdo S_4), luego se evalúa cual es la mejor de todas y en ese momento se completa un ciclo de ejecución o generación, reenviando la información de población más apta a todos los agentes (estado S_5). La labor del agente maestro termina al cumplir un número determinado de generaciones (estado S_6) dando como resultado final un recorrido valido que encuentra una solución cercana al mínimo global del PAV y en el mejor de los casos el mínimo global. Al terminar el agente efectúa operaciones de limpieza propias del sistema que se ejecutan cuando un agente ha terminado su tarea[28].

4.2.2 Agente esclavo.

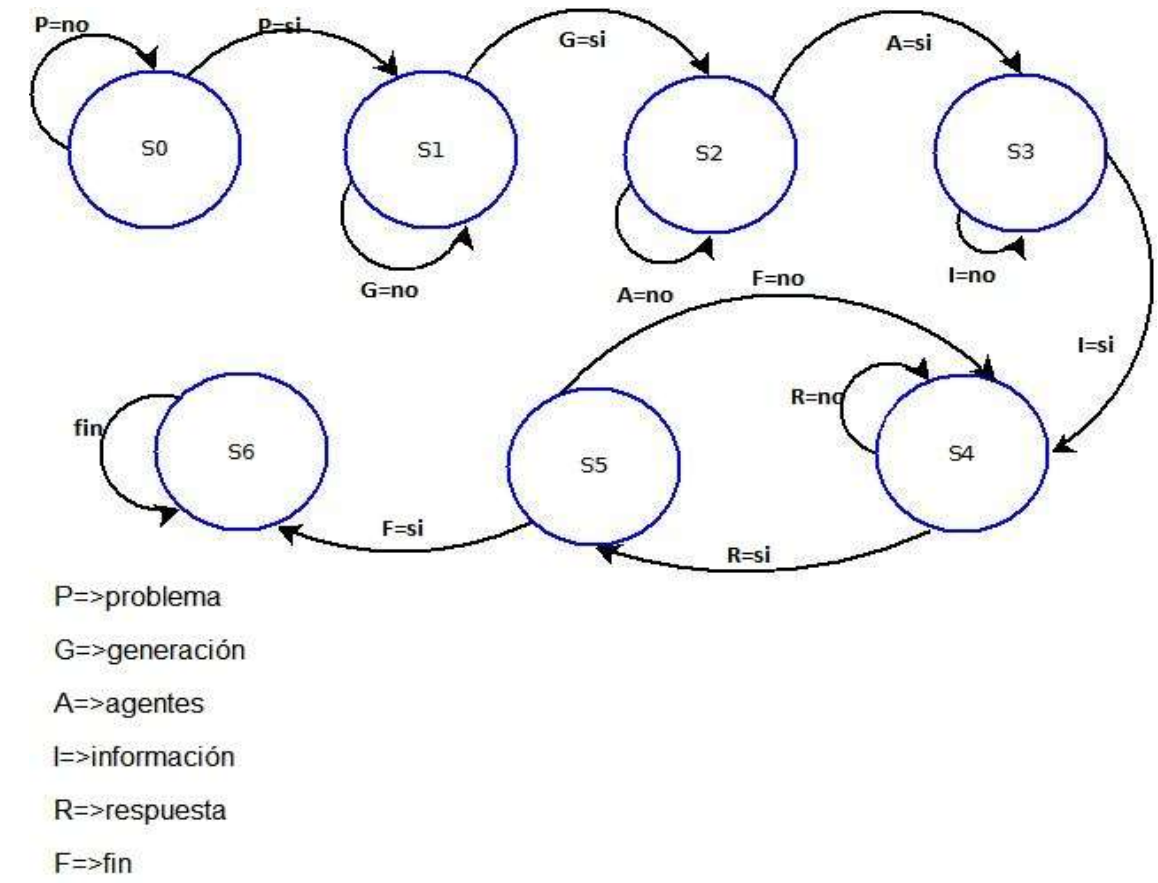
El agente esclavo por su parte al iniciar se registra en el DF como proveedor de servicios para solucionar el PAV (estado S_0) y queda a la espera de recibir la secuencia de datos enviados por algún agente maestro (estados S_1-S_5). Al recibir la información requerida (coordenadas de las ciudades, número de ciudades y número de pobladores del algoritmo genético), se asigna de forma aleatoria s una probabilidad de cruce (P_c) y una probabilidad(P_m) de mutación y realiza todo

[27] El Df es un servicio que provee el framwork JADE que facilita la comunicación entre agentes, permitiendo la búsqueda y registro de agentes sobre la plataforma.
[28] Ver ANEXO B

el proceso de crear una nueva generación (estado S_6), a continuación hace las operaciones de selección, cruce y mutación para encontrar individuos mejor adaptados y el resultado lo envía de regreso al agente maestro (estado S_7), al terminar con esta tarea queda a la espera de tres opciones: orden de parada, el retorno de otra población por parte del maestro u otro problema a resolver por parte de otro agente maestro (estados S_8-S_9) habiendo terminado el total de generaciones solicitadas por el usuario (estado S^{10})[29].

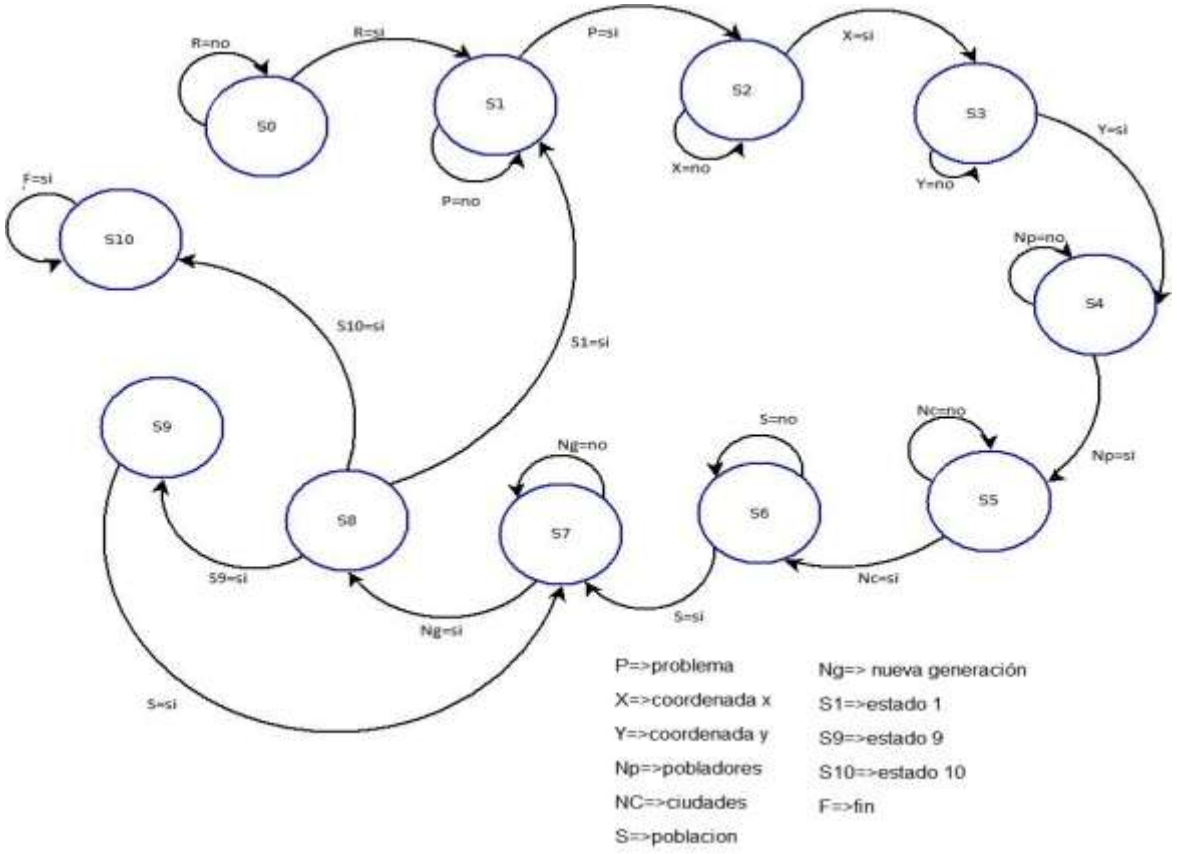
Figura 5 FSM del agente maestro software.

\



[29] Ver ANEXO C

Figura 6 FSM agente esclavo.



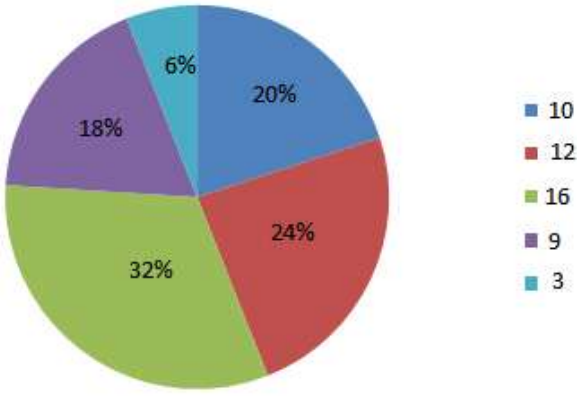
4.2.2.1 Operación de selección.

El proceso de selección[30] se realiza con base en el método de la ruleta o método Montecarlo, explicado en el capítulo 3, el procedimiento es sencillo y dado que no requiere que los datos tengan un ordenamiento, (cualidad que no es siempre muy fácil de realizar en hardware), trae consigo otros inconvenientes que reducen un poco la efectividad del algoritmo ya que un individuo que tenga una adaptación pobre puede llegar a ser elegido más de una vez para reproducción, otro

[30] Existen muchas formas de seleccionar los padres de una nueva generación el método d la ruleta y el método de selección por torneo son algunos de ellos.

inconveniente es que para la selección se requiere un tiempo de procesamiento proporcional al tamaño de la población, es decir, entre más grande la población, más tiempo requiere para realizar una selección de individuos a reproducirse, e.g. teniendo una población de 5 individuos donde la función de ajuste tiene por resultado el vector [10 12 16 9 3], la ruleta queda distribuida de la siguiente manera [10 22 38 47 50]. El valor de cada posición de la ruleta es el resultado de la suma de los valores de ajuste anteriores a ella, (en la ruleta la tercera posición es 38 resultado de sumar 10+12+16) mostrado gráficamente en la **figura 7**.

Figura 7 Probabilidades de selección usando el método de la ruleta.



4.2.2.2 Operación de cruce.

La operación de cruce[31] [32]implementada está basada en el cruce de dos puntos y funciona tomando dos puntos aleatorios dentro del rango de los padres, p_0 y p_1, se se extrae el segmento correspondiente del padre P_0 y se coloca al final del padre P_1, en un vector auxiliar Oc1, a continuación se eliminan los datos repetidos en este último, quedando de la combinación un nuevo hijo H_1, la probabilidad que

[31] En la operacion de cruce se debe tener en cuent las restricciones del problema para no generar individuos que no sean válido al problema en cuentión.
[32] Existen diversos métodos para realizer la operacion de cruce, en este trabajo solo se emlea el metodo referido arriba.

dos individuos se crucen para formar nuevos individuos suele estar alrededor del 90%.

En el ejemplo de la **tabla 3**, se muestra el cruce de los padres P_0 y P_1 en los puntos de cruce elegidos de forma aleatoria, para el ejemplo serán las posiciones 3 y 5 lo cual hace que el segmento de cruce sea la ristra 654 en el padre P_0.

Tabla 3 Operación de cruce entre dos individuos.

PC	1	2	3	4	5	6	7	A	B	C
P0	2	3	6	5	4	1	7			
P1	4	7	2	5	1	3	6			
Oc1	4	7	2	5	1	3	6	6	5	4
H1	7	2	1	3	6	5	4			

En el resultado se puede observar los genes heredados de los padres en el hijo y los cambios realizados en la estructura genética del mismo.

4.2.2.3 Operación de mutación.

En el proceso de mutación se elige de forma aleatoria dos puntos dentro del individuo y se intercambia el valor de sus posiciones, este proceso se realiza con una probabilidad baja que está entre el 1% y el 15%. En la **tabla 4**, se muestra el proceso de mutacion aplicado a un individuo.

Tabla 4 Operación de mutación de un individuo.

H1	1	2	3	4	5
H1*	1	5	3	4	2

4.2.2.4 Codificación.

En el algoritmo genético se está implementando una codificación decimal entera directa, donde cada componente del individuo representa una ciudad a recorrer en el orden establecido por el orden de los números, así un individuo formado por la cadena [1 3 2 4 6 5] irá de la ciudad uno a la tres, seguidamente de la tres a la dos, luego de la dos a la cuatro y así sucesivamente hasta terminar todo el recorrido.

Recordemos que en el algoritmo genético cada individuo representa una posible solución al problema y que una población de individuos corresponde a múltiples soluciones.

4.2.2.5 Fitnes.

La función de ajuste implementada es de tipo estandarizada con la forma:

$$Fit = K_{MAX} - d$$

$$\boldsymbol{Fit = kmax - \sum_{i=0}^{N-1} d(v_i, v_{i+1}) + d(v_1, v_n)}$$

Donde

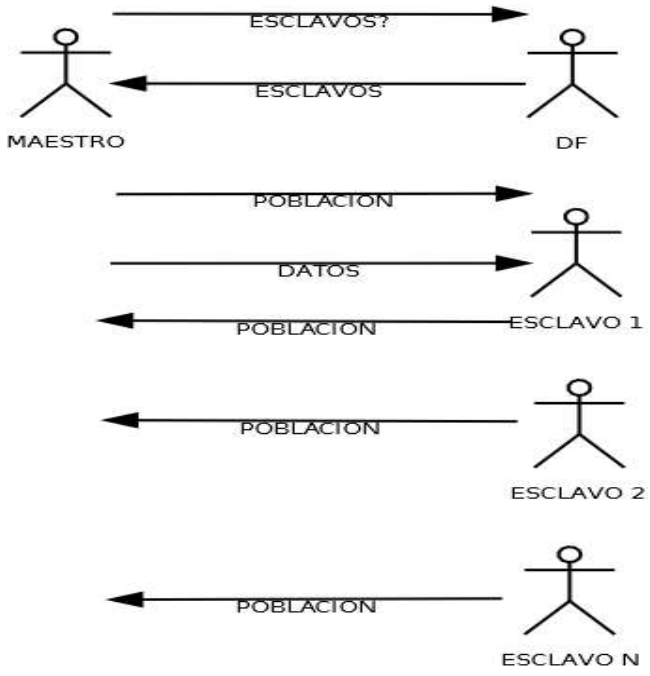
$$\boldsymbol{d = \sqrt[2]{(X_i - X_{i+1})^2 + (Y_i - Y_{i+1})^2}}$$

Teniendo en cuenta que se desea maximizar la función *F* disminuyendo la distancia *d* entonces k_{max} debe ser una constante relativamente grande correspondiente al problema a solucionar.

4.3 SISTEMA MULTIAGENTE.[33]

El sistema multiagente conformado por un agente maestro y uno o varios agentes esclavo se implementa usando la plataforma JADE, como se mencionó anteriormente la comunicación entre los agentes se hace usando las clases que esta plataforma proporciona siguiendo el estándar FIPA, por medio del paso de mensajes (**Figura 8**), los agentes que se encuentren en la plataforma se registran en el directorio de páginas amarillas o directorio facilitador (DF), informado cuáles son sus cualidades, el nombre y la dirección de red en que puede ser encontrados. A este directorio se registran los agentes que puedan proveer uno o múltiples servicios y que estén disponibles para su uso. En el caso del sistema multiagente propuesto en este trabajo son los agentes esclavo quienes se registran en el DF y el agente maestro es quien consulta qué agentes hay disponible para su uso.

Figura 8 Interacciones entre agentes.



[33] Ver ANEXO Ver ANEXO J

Una vez encontrado al menos un agente esclavo disponible en el DF la comunicación se realiza entre maestro y esclavos de forma transparente.

Al interior de la clase maestro se implementaron diversas subclases y métodos para la inicialización de las poblaciones además de la coordinación entre los agentes y toda la lógica requerida para la ejecución del algoritmo genético paralelo maestro esclavo de forma distribuida. Se debe tener en cuenta que las clases maestro y esclavo heredan de la clase superior *Agent* todas las propiedades que ser agente conlleva y que éstas, las clases maestro y esclavo, solo implementan la lógica requerida para la solución del problema.

4.4 ANÁLISIS DEL SISTEMA MULTIAGENTE SOFTWARE PARA SOLUCIONAR EL PROBLEMA DEL AGENTE VIAJERO.

El análisis del sistema multiagente dentro de las metodologías de desarrollo software consta de los siguientes ítems:

- Modelo de agentes, se identifican los agentes que hacen parte del sistema, con que agentes deben relacionarse y de qué forma, cuáles son los actores y los papeles que desempeñan en el sistema.
- Modelo de interacciones, representa la forma en que los diferentes agentes se comunican, ante qué mensajes debe reaccionar y cuál es el formato de los mismos.

4.4.1 Actores y papeles.

Agente maestro

- **Función:** cargar los datos requeridos para la solución de un problema específico del PAV, es el responsable de generar la población inicial, buscar los agentes esclavo en el directorio de agentes, enviar la población inicial,

esperar, recibir , evaluar las poblaciones provenientes de los esclavos y determinar cuál es el mejor individuo para dar una respuesta al usuario.

- **Capacidades:** encontrar una solución a un problema específico PAV.

- **Comunicación:** envío y recepción de mensajes ACL dentro de la plataforma de agentes JADE.

Agente esclavo

- **Función:** realizar las operaciones genéticas sobre la población proporcionada por el agente esclavo y devolver la nueva generación de pobladores para ser evaluados por el agente maestro.
- **Capacidades:** encuentra una nueva generación de individuos a partir de una proporcionada, teniendo en cuenta los operadores genéticos de cruce y mutación conservando las restricciones del PAV.
- **Comunicación:** se registra en el directorio de agentes proporcionando el servicio del PAV, se comunica con el agente maestro por medio de mensajes ACL dentro de la plataforma JADE.

4.4.2 Casos de uso.

En la **tabla 5** y **figura 9,** se presentan los casos de uso del sistema multiagente como parte de la documentación metodológica.

Tabla 5 Casos de uso del SMA.

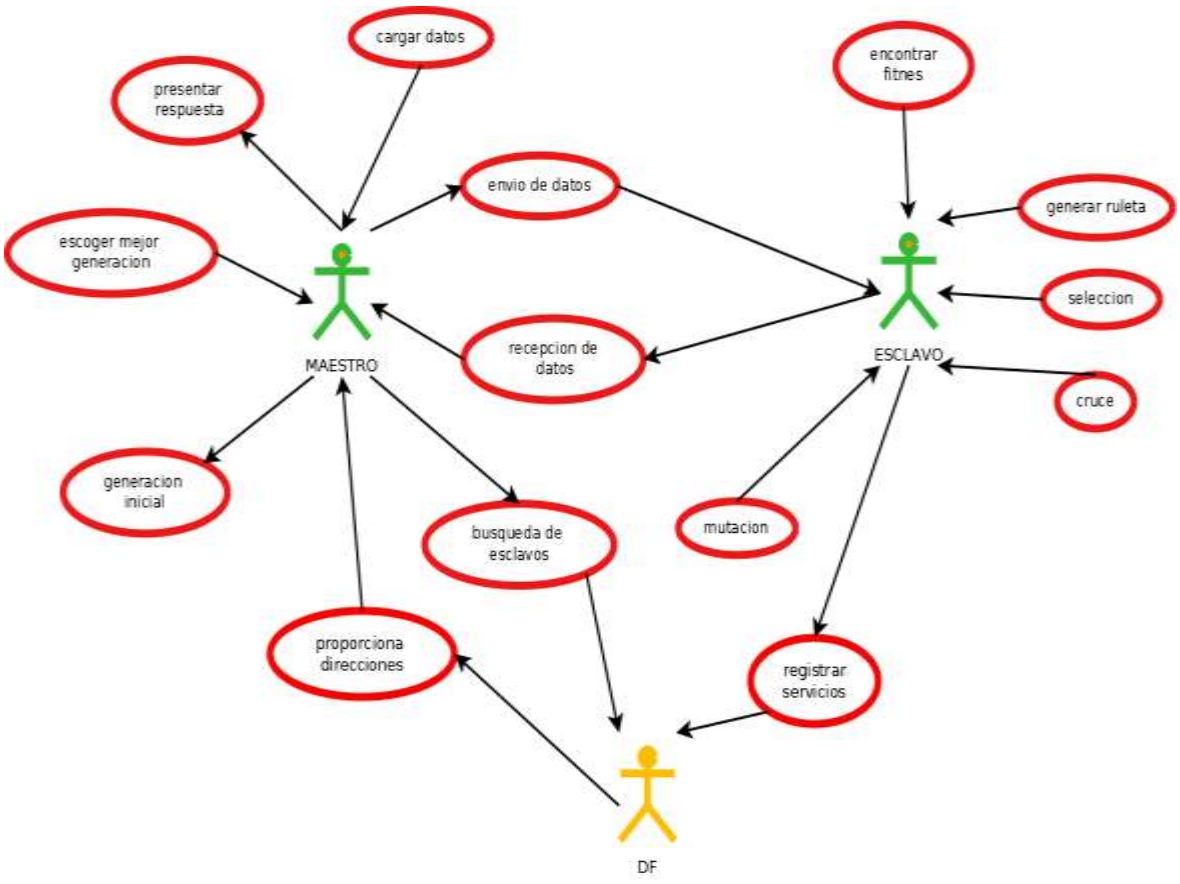
Caso de uso	Cargar datos
Actor	Agente maestro
Descripción	El agente maestro extrae los datos del problema a solucionar

Caso de uso	Presentar respuesta
Actor	Agente maestro
Descripción	El agente maestro termina la ejecución mostrando en pantalla la solución obtenida e informando al agente esclavo que su tarea ha terminado.
Caso de uso	Seleccionar mejor población
Actor	Agente maestro
Descripción	De las respuestas recibidas escoge la que tenga el mejor ajuste para reenviar los datos a los esclavos para iniciar una nueva generación.
Caso de uso	Cargar datos
Actor	Agente maestro
Descripción	El agente maestro extrae los datos del problema a solucionar
Caso de uso	Generación inicial
Actor	Agente maestro
Descripción	El agente maestro a partir de los datos del problema crea de forma aleatoria una población de individuos para inicialización del algoritmo genético.
Caso de uso	Envío de datos
Actor	Agente maestro, agentes esclavo
Descripción	El agente maestro envía la mejor generación a los agentes esclavo mediante un mensaje INFORM
Caso de uso	Recepción de datos
Actor	Agente maestro, agentes esclavo

Descripción	El agente maestro recibe las poblaciones con nuevos individuos de los agentes esclavo
Caso de uso	Búsqueda de esclavos
Actor	Agente maestro, DF
Descripción	El agente maestro consulta el agente DF en busca de agentes esclavo que puedan proporcionar el servicio requerido PAV.
Caso de uso	Mutación
Actor	Agente esclavo
Descripción	El agente esclavo realiza la operación de mutación sobre un individuo de la población seleccionado.
Caso de uso	Registrar servicios
Actor	Agente esclavo, DF
Descripción	El agente esclavo registra su dirección y servicios en el directorio
Caso de uso	Cruce
Actor	Agente esclavo
Descripción	El agente esclavo realiza la operación de mutación sobre 2 individuos seleccionados.
Caso de uso	Selección
Actor	Agente esclavo
Descripción	El agente esclavo selecciona de forma aleatoria 2 individuos de la población para reproducción.
Caso de uso	Generar ruleta
Actor	Agente esclavo

Descripción	El agente esclavo genera la ruleta con base en los valores de ajuste de los individuos de la población.
Caso de uso	Encontrar fitnes
Actor	Agente esclavo
Descripción	El agente esclavo encuentra el fitnes de la población recibida para realizar las operaciones genéticas con base en ello

Figura 9 Diagrama casos de uso del SMA.

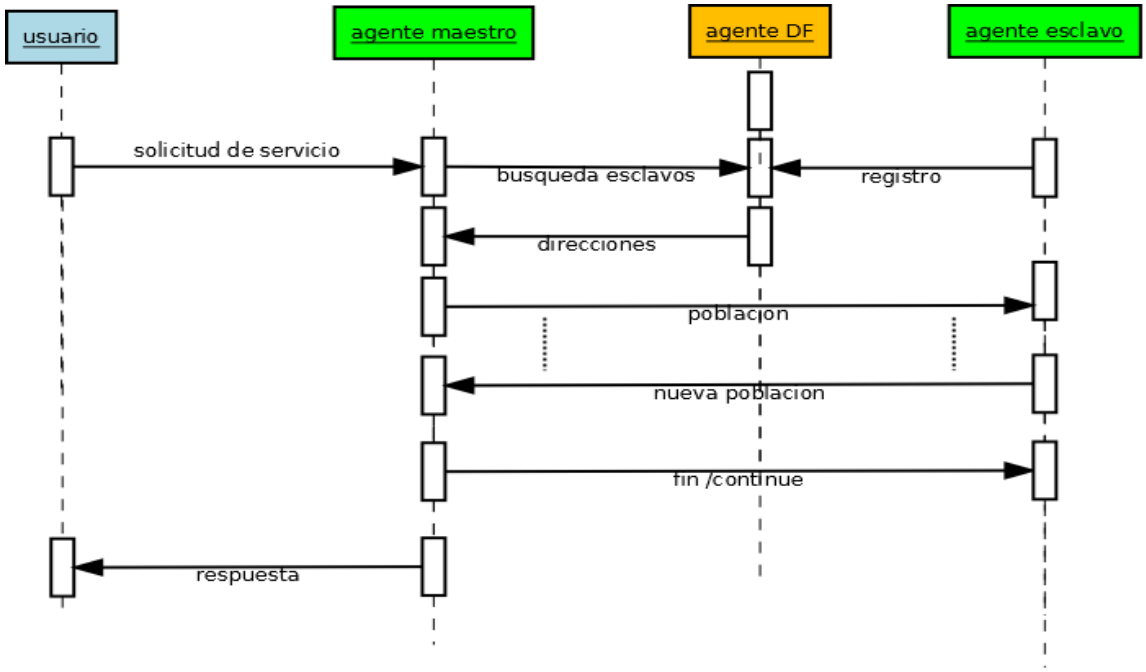


4.4.3 Diagrama de interacciones.

El modelo de interacciones representa de qué forma se comunican los agentes, cuáles son os mensajes a los que responden y de qué forma lo hacen (**figura 10**).

En esta implementación la comunicación se hace a través de la plataforma JADE haciendo uso del servicio de mensajería ACL que ésta provee[34].

Figura 10 Diagrama de interacciones.



4.4.4 Diagrama de clase.

El sistema está compuesto por dos subclases, la clase maestro y la clase esclavo que heredan de la clase *Agent*, la cual es la clase principal que pertenece a la

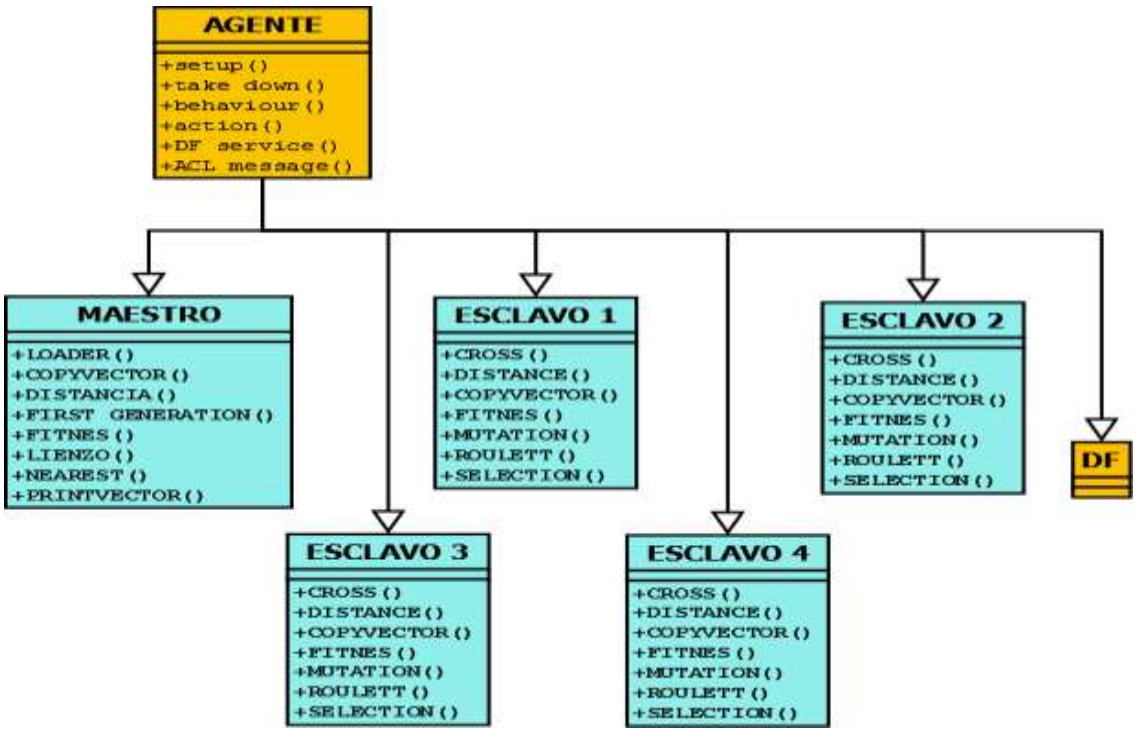
[34] Ver ANEXO D

plataforma JADE, las cuales poseen la lógica interna, sus estados mentales, sus planes de ejecución y la solución al problema en sí (**figura 11**).

La clase *Agent* posee los métodos de inicialización, ejecución y terminación de los agentes, es la que permite que los agentes vivan dentro de la plataforma JADE y realiza las acciones de planificación de los diferentes comportamientos de los agentes.

La clase maestro tiene como funciones principales las de leer los datos del problema, inicializar la población del algoritmo genético y evaluar las poblaciones recibidas de los agentes esclavo para determinar cuál es el mejor individuo y poder dar una solución aceptable al TSP. Debe recordarse que en sí cada una de estas clases al ser ejecutada es un agente que vive dentro de la plataforma JADE.

Figura 11 Diagrama de clase del SMA.

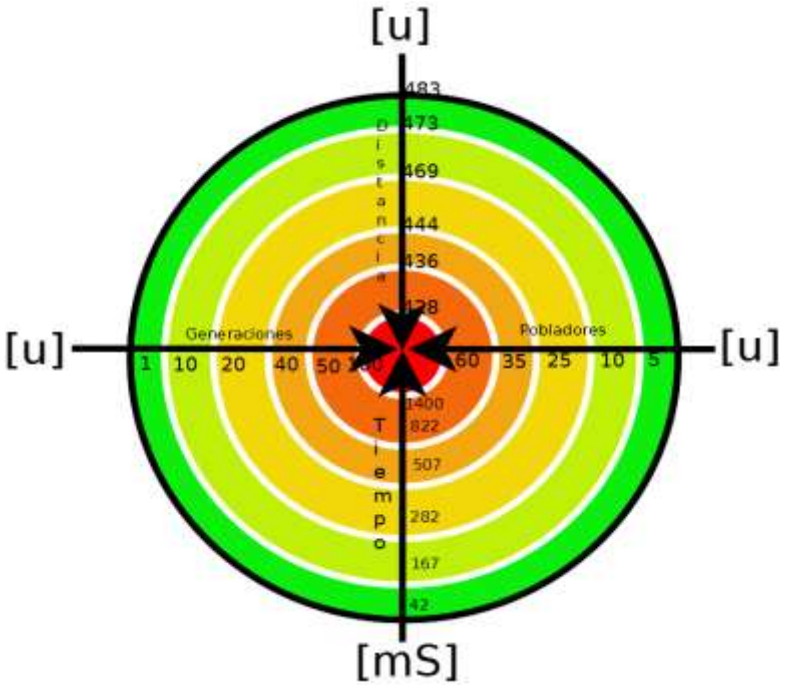


4.5 PRUEBAS Y RESULTADOS.

Las pruebas se efectuaron los problemas típicos del PAV tomados del TSPLIB, eil101, eil51, eil76 y oliv30. Las cuales se efectuaron con diferentes probabilidades de cruce y mutación en los agentes esclavo y con diferente número de esclavos, como referencia se compara contra algunos algoritmos heurísticos conocidos, teniend en cuenta en la comparación solo el resultado final y no el tiempo empleado en encontrar esta solución.

Al realizar variaciones en el numero de generaciones y de pobladores se obtiene que las dos tienen incidencia directa en la distancia encontrada y en el tiempo que el sistema se demora en dar una respuesta, en este aspecto para un problema como es el oliv30 se hizo un barrido de generaciones entre 1 y 100 y un barrido de pobladores entre 5 y 60, teniendo como resultados tiempos entre 42 ms y 1400ms y recorridos con distancias entre 483 y 421 (**Figura 12**).

Figura 12 Exactitud del SMA para el problema oliv30.



4.5.1 Prueba con el problema oliv30.

Para el problema oliv30 se empleó como configuración del sistema un agente maestro, cuatro agentes esclavo, 300 generaciones y 60 individuos en la población, tras ejecutar el algoritmo varias veces se obtuvieron los resultados presentados en **la tabla 6**.

Tabla 6 Pruebas realizadas al SMA software con el problema oliv30.

Ruta	D[35]	T[36]	G[37]	P[38]
30 29 27 26 28 25 24 23 22 21 17 20 18 19 16 15 14 13 12 11 6 10 9 8 7 5 4 3 2 1	426	2986	300	60
19 17 16 15 14 13 12 11 6 10 9 8 7 5 4 3 2 1 30 29 27 26 28 25 24 23 22 21 20 18	421	2904	300	60
19 17 16 15 14 13 12 11 6 10 9 8 7 5 4 3 2 1 30 29 28 27 26 25 24 23 22 21 20 18	422	2856	300	60
17 19 18 20 21 22 23 24 25 28 26 27 29 30 1 2 3 4 5 12 13 11 6 10 7 8 9 15 14 16	427	2853	300	60

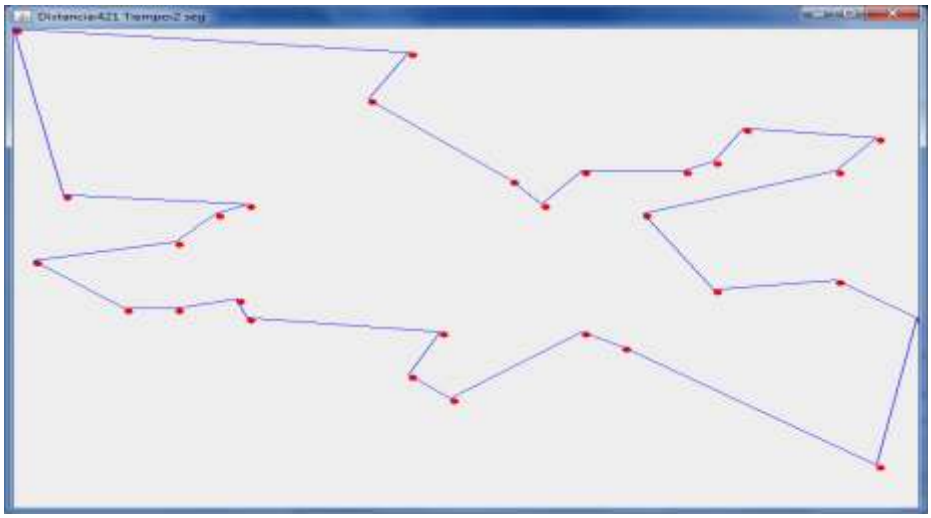
Ruta encontrada: 19 17 16 15 14 13 12 11 6 10 9 8 7 5 4 3 2 1 30 29 27 26 28 25 24 23 22 21 20 18.

Distancia = 421, Tiempo = 2904

Siendo ésta la solución óptima del problema, en la **figura 13** se observa el recorrido generado.

[35] Distancia total del tour.
[36] Tiempo que se demora el SMA en dar una solución dado en mS.
[37] Numero de generaciones empleadas.
[38] Numero de pobaldores que componen una generaion.

Figura 13 Recorrido óptimo oliv30.



Los resultados de solucionar el problema **oliv30** con diferentes algoritmos heurísticos se presenta en la **tabla 7**.

Tabla 7 solución al problema oliv30 con otros algoritmos heurísticos[39].

Greedy	482
Boruvka	526
Qboruvka	512
Nearest N.	546
Ramdom	1343

Mediante una serie de pruebas realizadas con el SMA se observan los errores relativos porcentuales mostrando que las respuestas obtenidas se encuentran por arriba del 95% de efectividad con realcion a la mejor solución conocida del problema. Cosnstatando que las soluciones encontradas por el SMA son buenas soluciones, como se observa en la **tabla 8**.

[39] Estas soluciones se obtienen con las herramientas proporcionadas por el software Concord.

Tabla 8 Error relativo de la solución al problema oliv30 con el SMA software.

	Distancia	Solucion	Error relativo Erp	100-Erp
	430	421	2,13776722	97,8622328
	442	421	4,98812352	95,0118765
	437	421	3,80047506	96,1995249
	421	421	0	100
	430	421	2,13776722	97,8622328
	422	421	0,23752969	99,7624703
	422	421	0,23752969	99,7624703
	442	421	4,98812352	95,0118765
	430	421	2,13776722	97,8622328
	422	421	0,23752969	99,7624703
	442	421	4,98812352	95,0118765
Promedio	430,991	421	2,3537033	97,6462967

4.5.2 Prueba con el problema eil51.

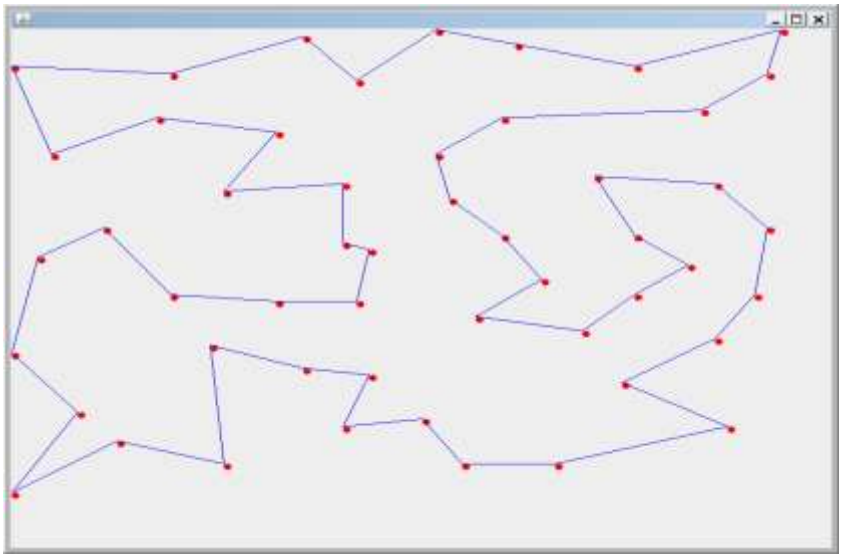
En la **tabla 9** se muestran los resultados obtenidos al resolver el problema eil51 tras ser ejecutado varias veces.

Tabla 9 Pruebas del SMA software con el problema eIL51.

Ruta	D	T	G	P
1 32 11 38 5 49 9 50 16 29 21 34 30 10 39 33 45 15 44 37 17 47 6 23 7 43 24 14 25 18 13 41 40 19 42 4 12 46 51 27 48 8 26 31 28 3 36 35 20 2 22	449	8311	500	102
27 48 8 26 31 28 3 36 35 20 29 21 16 2 22 1 32 11 38 5 49 9 50 34 30 10 39 33 45 15 44 37 17 18 6 23 7 43 24 14 25 13 41 40 19 42 4 47 12 46 51	441	6233	500	102
18 47 12 46 51 27 48 6 24 43 23 7 26 8 31 28 3 36 35 20 2 22 1 32 11 38 5 49 9 50 16 29 21 34 30 10 39 33 45 15 44 37 17 4 42 19 40 41 13 25 14	437	6251	500	102
18 47 12 46 51 27 6 48 23 24 43 7 26 8 31 28 3 36 35 20 2 22 1 32 11 38 5 49 9 50 16 29 21 34 30 10 39 33 45 15 44 37 17 4 42 19 40 41 13 25 14	431	6063	500	102

Siendo la solución óptima de 426 la más próxima encontrada (**figura 14**), por medio del sistema multiagente es de 431.

Figura 14 Mejor recorrido encontrado con el SMA eil51



Al solucionar el problema eil51 con diferentes algoritmos heurísticos se obtienen los datos de la **tabla 10**.

Tabla 10 Solución al problema EIL51 con otros algoritmos heurísticos.

Greedy	521
Boruvka	545
Qboruvka	480
Nearest N.	499
Ramdom	1761

En las pruebas realizadas al sistema con el problema eil51 se tienen que los errores realtivos porcentuales están por abajo del 10% (**tabla 11**), teniendo en cuenta que al aunmentar el tamaño del problema, la dificultad en encontrar respuestas satistfactorias también lo hace en forma exponencial, con respecto de

los algoritmos heurísticos de prueba sigue teniendo respuestas en el peor de los casos hasta un 2.8% mejores.

Tabla 11 Error relativo de las solución al problema EIL51 con el SMA software.

	Distancia	Solucion	Error relativo Erp	100-Erp
	431	426	1,17370892	98,8262911
	438	426	2,81690141	97,1830986
	437	426	2,58215962	97,4178404
	443	426	3,99061033	96,0093897
	459	426	7,74647887	92,2535211
	446	426	4,69483568	95,3051643
	447	426	4,92957746	95,0704225
	450	426	5,63380282	94,3661972
	454	426	6,57276995	93,42723
	449	426	5,39906103	94,600939
	459	426	7,74647887	92,2535211
Promedio	452,636364	426	4,84421682	95,1557832

4.5.3 Prueba con el problema eil76.

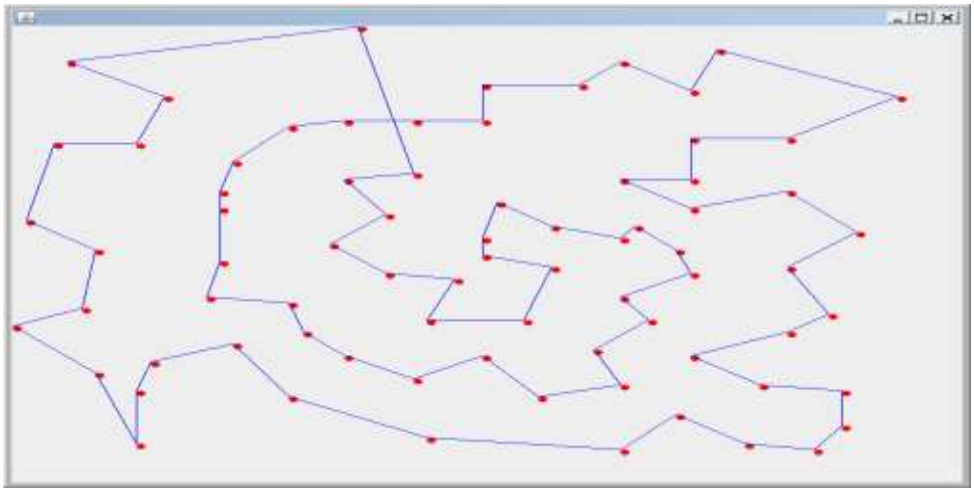
La **tabla 12**, representa las soluciones obtenidas al correr el SMA para el problema eil76.

Tabla 12 Pruebas del SMA software con el problema eil76.

Ruta	D	T	G	P
12 40 17 51 6 68 75 76 26 67 4 34 46 52 27 45 29 48 47 36 69 71 60 70 20 37 5 15 57 13 54 19 8 35 7 53 14 59 11 66 65 38 72 39 9 32 44 3 16 63 33 73 1 43 41 42 64 22 62 28 74 2 30 21 61 56 23 49 24 18 50 25 55 31 10 58	572	12167	300	102
4 67 34 46 8 35 7 58 72 39 9 32 44 3 16 33 63 23 56 49 24 18 50 25 55 31 10 38 65 66 59 11 53 14 19 54 13 27 52 45 29 48 47 21 74 28 62 73 1 43 41 42 64 22 61 69 36 71 60 70 20 37 57 15 5 30 2 68 6 51 17 40 12 26 76 75	567	5878	300	102
6 68 75 76 26 67 4 34 46 52 27 45 29 48 47 36 69 71 60 70 20 37 5 15 57 13 54 19 8 35 7 53 14 59 11 66 65 38 10 58 72 39 9 32 44 3 16 63 33 73 2 30 74 21 61 22 28 62 1 43 42 64 41 56 23 49 24 18 50 25 55 31 12 40 17 51	556	12510	300	102
75 76 26 67 34 46 52 27 45 29 48 47 21 74 28 62 73 33 63 16 3 44 32 9 39 72 58 10 38 65 11 66 59 14 53 35 7 8 19 54 13 57 15 5 37 20 70 60 71 36 69 61 22 1 43 42 64 41 56 23 49 24 18 50 25 55 31 12 40 17 51 6 68 2 30 4	553	17456	300	102

Siendo la solución óptima de 538, la más próxima encontrada (**figura 15**), por medio del sistema multiagente es de 553.

Figura 15 Mejor ruta encontrada con el SMA eil76



El mismo problema resuelto con diferentes algoritmos heurísticos se presenta en la **tabla 13**.

Tabla 13 Solución al problema eil76 con otros algoritmos heurísticos.

Greedy	631
Boruvka	574
Qboruvka	609
Nearest N.	661
Ramdom	2342

Para el problema de 76 ciudades el error relativo porcentual se presenta por debajo del 8% con respecto a la solución óptima conocida (**tabla 14**). En el peor de los casos se encuentra un 1% por debajo de las solución del algoritmo heurístico Boruvka, sin embargo en el promedio de las soluciones tras 10 iteraciones se encuentra un 1% por arriba de éste.

Tabla 14 Error relativo de la solución al problema eil76 con el SMA software

	Distancia	Solucion	Error relativo Erp	100-Erp
	575	538	6,87732342	93,1226766
	567	538	5,39033457	94,6096654
	579	538	7,62081784	92,3791822
	553	538	2,78810409	97,2118959
	566	538	5,20446097	94,795539
	564	538	4,83271375	95,1672862
	569	538	5,76208178	94,2379182
	565	538	5,01858736	94,9814126
	576	538	7,06319703	92,936803
	565	538	5,01858736	94,9814126
	571	538	6,133829	93,866171
Promedio	568,181818	538	5,61000338	94,3899966

4.5.4 Prueba con el problema eil101.

El problema de 101 cuidades del TSPLIB solucionado con el SMA se presenta en la **tabla 15**.

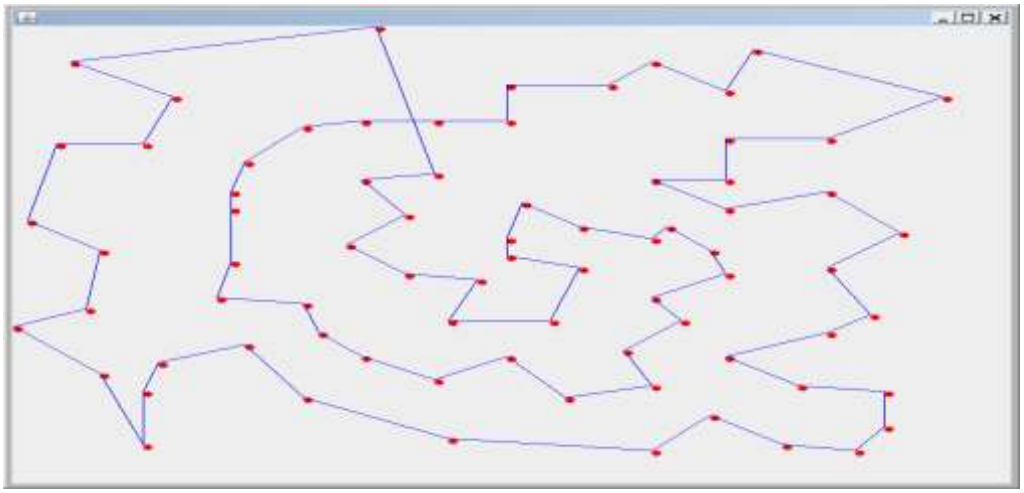
Tabla 15 Pruebas del SMA software con el problema eil101.

Ruta	D	T	G	P
67 23 56 75 74 22 41 72 73 21 40 58 53 101 27 28 26 12 80 68 77 3 79 78 34 81 33 51 9 35 71 65 66 20 30 70 31 88 62 10 90 32 63 11 19 48 82 8 45 17 84 5 60 83 18 89 6 94 95 97 92 59 99 96 93 98 37 100 91 85 61 16 44 14 42 87 13 2 57 15 43 38 86 46 47 36 49 64 7 52 69 1 50 76 29 24 54 55 25 4 39	688	14249	300	102
42 87 97 95 94 6 96 99 59 92 98 37 100 91 85 93 61 16 86 44 14 38 43 15 57 2 13 58 40 53 101 27 28 26 12 80 68 77 3 79 78 34 81 33 51 9 71 35 65 66 20 30 70 31 88 7 82 48 47 36 49 64 11 62 10 32 90 63 19 46 8 45 17 84 5 60 83 18 89 52 69 1 50 76 29 24 54 55 25 4 56 75 23 67 39 72 21 73 74 22 41	680	14127	300	102
39 67 23 56 75 41 22 74 72 73 21 40 58 53 101 27 28 26 12 80 68 77 3 79 78 34 81 33 51 9 71 35 65 66 20 30 70 31 88 62 10 32 90 63 11 19 48 82 7 52 89 6 94 95 97 92 59 96 99 93 98 37 100 91 85 61 16 44 14 42 87 13 2 57 15 43 38 86 17 84 5 60 83 18 8 45 46 47 36 49 64 69 1 50 76 29 24 54 55 25 4	678	39302	300	202

26 2 57 15 43 38 86 17 84 5 60 83 18 8 45 46 47 36 49 64 63 90 32 10 31 88 62 11 19 48 82 7 52 89 6 94 95 97 92 59 96 99 93 98 37 100 91 85 61 16 44 14 42 87 13 58 40 21 73 72 74 22 41 75 56 39 23 67 25 55 54 4 24 29 80 68 77 3 79 78 34 81 33 51 9 35 71 65 66 20 30 70 1 69 27 101 53 28 50 76 12	667	30334	300	202

Siendo las solución óptima de 629, la más próxima encontrada (**figura 16**), por medio del sistema multiagente es de 667.

Figura 16 Mejor ruta encontrada con el SMA eil101.



Para el mismo problema solucionado con diferentes algoritmos heurísticos se obtuvieron los datos presentados en la **Tabla 16.**

Tabla 16 Solución al problema eil101 con otros algoritmos heurísticos.

Greedy	769
Boruvka	711
Qboruvka	746
Nearest N.	784
Ramdom	3459

En el problema de 101 ciudades se el error relativo (**tabla 17**), se encuentra por debajo del 12% con respecto a la solución optima y comparado con los algoritmos heurísticos en el peor de los casos se encuentra hasta un 1.7% por encima.

Tabla 17 Error relativo de la solución al problema eil101 con el SMA software.

	Distancia	Solucion	Error relativo Erp	100-Erp
	685	629	8,903020668	91,0969793
	688	629	9,379968203	90,6200318
	698	629	10,96979332	89,0302067
	695	629	10,49284579	89,5071542
	699	629	11,12877583	88,8712242
	691	629	9,856915739	90,1430843
	685	629	8,903020668	91,0969793
	691	629	9,856915739	90,1430843
	682	629	8,426073132	91,5739269
	694	629	10,33386328	89,6661367
	676	629	7,47217806	92,5278219
Promedio	689,454545	629	9,611215494	90,3887845

Al observar los resultados obtenidos se puede decir que la implementación propuesta proporciona buenos resultados en comparación con algunos algoritmos heurísticos proporcionados en el software *Concord* y en el caso de del oliv30 se ha encontrado la solución óptima conocida.

Debido a que la plataforma JADE es muy costosa computacionalmente[40], se observa que los tiempos de ejecución son bastante grandes, llegando a demorarse para el problema oliv30 que es relativamente pequeño (30 ciudades) hasta 3.5 segundos en proporcionar una respuesta en un computador con un procesador *Intel core* I5 de 4 núcleos a 2.5Ghz, tiempo durante el cual el procesador llega a su máximo de uso de los 4 núcleos evidenciando la gran

[40] Ver ANEXO E

demanda de recursos que exigen este tipo de sistemas implementados en software.

CAPÍTULO 5

IMPLEMENTACION HARDWARE: SISTEMA MULTIAGENTE PARA SOLUCIONAR EL PROBLEMA DEL AGENTE VIAJERO.

La implementación hardware del sistema multiagente para solucionar el problema del agente viajero, se realizó con base en la implementación software siguiendo los mismos pasos de diseño que se siguieron con esta, procurando que las las dos fuesen similares en la forma de resolver el problema es decir aplicando los mismos algoritmos de solución.

5.1 HERRAMIENTAS DE DESARROLLO.

Para el desarrollo del sistema multiagente hardware se emplearon las herramientas:

- QUARTUS II v11.0 Web edition.
- DIA v 0.9
- MOdelSim-Altera V10.0.
- CONCORD v1.1
- Herrameineta JAVA para grafica de resultados hardware.

5.2 UNIDADES FUNCIONALES

Las unidades funcionales o entidades básicas son las que componen de forma operativa el sistema, es decir, las que realizan las operaciones básicas dentro de la ejecución del algoritmo, entre las que se encuentran sumadores, multiplicadores, restadores, raíz cuadrada, potenciación, etc. Estas funciones fueron implementadas por medio de las megas funciones proporcionadas por QUARTUS II.

Se aclara que debido a la limitación de recursos al implementar un sistema sobre FPGA el problema se limita a una cantidad máxima de 101 ciudades, es decir, las pruebas se realizan con problemas hasta de 101 ciudades y poblaciones máximas de 100 individuos en donde los datos de las componentes cartesianas no superen los 7 bits.

5.3 AGENTE HARDWARE.

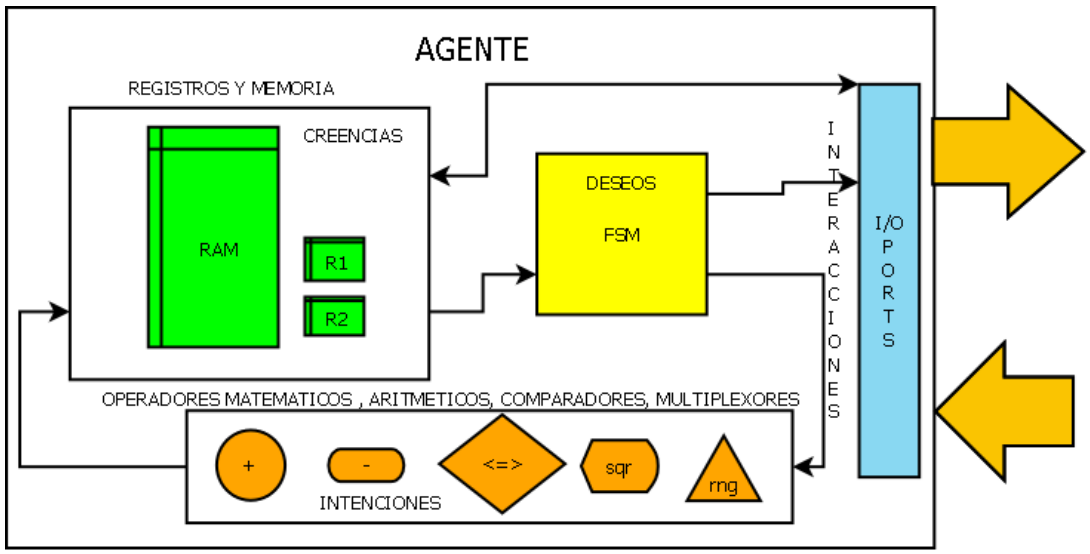
Debido a que no hay mucha información acerca de la creación y diseño de agentes hardware, en este trabajo se hace uso de una arquitectura BDI modificada para tal fin, asumiendo que en el hardware las creencias, los deseos y las intenciones hacen parte del mismo, obteniéndose una aproximación al modelo de la siguiente forma[16][17][18].

- **Creencias:** son el conocimiento que el agente tienen del ambiente, lo que sabe, en este caso se asume como los bancos de memoria y registros donde se guarda la información del problema, distancias, individuos, población etc.
- **Deseos:** es lo que el agente puede hacer para modificar y percibir el ambiente, como puede hacer uso del conocimiento, en este caso se refiere a la capacidad funcional que tienen el hardware, sumadores multiplicadores, multiplexores que en conjunto pueden llevar a cabo las tareas en cada estado mental del agente.

- **Intenciones:** se refiere a lo que el agente quiere hacer y cuáles son los pasos que debe seguir para conseguir su finalidad, resolver el problema del agente viajero, pudiendo así asemejar éstas a las máquinas de estado finito, que hacen que las creencias sean modificadas por los deseos siguiendo las intenciones del agente.
- **El modelo de agente**: se basa en los deseos las creencias y las intenciones.
- **El modelo de interacciones**: que es la forma, protocolo y formato que se tienen al comunicarse con otros agentes.

Tras realizar esta abstracción de lo que es una arquitectura para agentes software y aplicándola a una implementación hardware (**Figura 17**), se propone entonces para el diseño del sistema empezar modelando los deseos y creencias y por ultimo las intenciones.

Figura 17 Abstracción hardware de un agente bajo el modelo BDI.

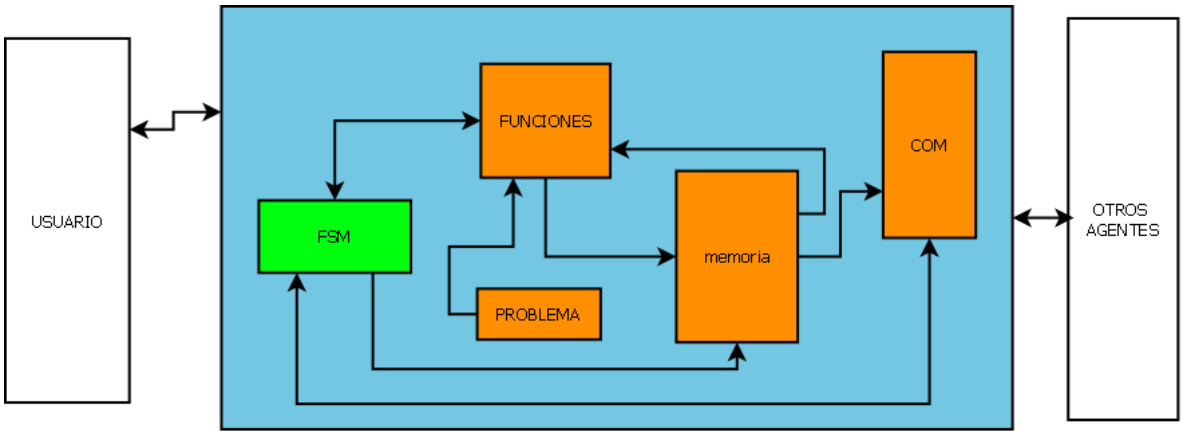


5.3.1 Agente maestro hardware.

Teniendo en cuenta las operaciones de inicialización, comunicación y selección de la mejor solución entre un conjunto de soluciones o población, además de las cualidades que debe poseer un agente (veracidad, autonomía, sociabilidad y

benevolencia) se ven reflejadas en el diseño propuesto, debido a que el agente tiene capacidad de comunicación con el usuario y con otros agentes, es autónomo en la selección de parámetros tales como el porcentaje de mutación, el porcentaje de cruce y la selección de la semilla del generador de números pseudo aleatorio, lo cual le permite tomar una ruta de búsqueda según su propio deseo y no está condicionado en este aspecto a la intervención humana o de otros agentes (**figura 18**). La información que comunica siempre es la que cree que es la más acertada según las condiciones de diseño, para el caso la mejor población encontrada y la mejor solución encontrada. Siendo éstas las características principales de un agente.

Figura 18 Esquema básico del agente maestro hardare.



5.3.1.1 Modelo de creencias.

Las creencias del agente maestro hardware están dadas por la información que maneja del ambiente en que se mueve, en este caso se habla del problema del agente viajero y de un algoritmo genético (AG), así que la información está dada por poblaciones, individuos, función de ajuste, padres e hijos. Siendo estas creencias la memoria y los registros dedicados dentro del circuito para contener todas estas características del entorno (PAV, AG).

Aprovechando el paralelismo implícito que presentan las implementaciones hardware el agente maestro está en capacidad de:

- Inicializar de forma paralela hasta cuatro diferentes poblaciones.
- Seleccionar la mejor población y enviarla a cuatro agentes esclavo,
- Recibir las nuevas poblaciones.
- Calcular cual es la mejor de las poblaciones recibidas.
- Reiniciar el ciclo hasta terminar la tarea dada por el número de generaciones máxima.
- Devolver el valor de la mejor solución encontrada y la ruta a seguir por el comerciante.

5.3.1.2 Modelo de intenciones.

Para las intenciones, el agente está en capacidad de realizar las operaciones de generación de números pseudo aleatorios, encontrar la distancia euclidiana entre dos puntos, la distancia total de un recorrido y poder realizar operaciones de copiado, concatenación y comparación de vectores.

5.3.1.3 Modelo de deseos.

En este aspecto y haciendo uso de las intenciones, el agente está en capacidad de comunicarse de forma eficiente con otros agentes, determinar la presencia y/o ausencia de agentes esclavo, ajustar el tamaño de memoria requerido para realizar las operaciones, identificar individuos independientes dentro de la población, encontrar las bondades de éstos y determinar cuál es el mejor de ellos.

5.3.1.4 Modelo del agente.

El agente maestro está compuesto por cuatro módulos independientes, cada uno con capacidad de recibir una población, de encontrar cual es la mejor entre dichas

poblaciones y enviarla a los agentes esclavo para que con base en esta generen una nueva.

5.3.1.5 Modelo de interacciones.

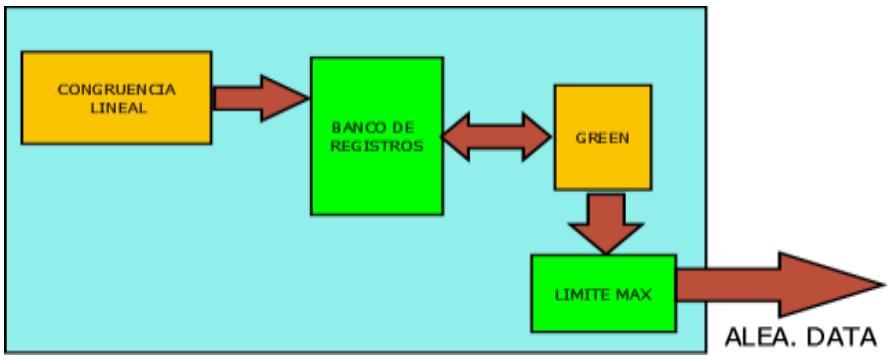
El agente maestro se puede comunicar con el usuario recibiendo la información esencial para solucionar el problema, y poder dar una respuesta válida. Del mismo modo se puede comunicar hasta con cuatro agentes esclavo enviándoles la población inicial para que estos puedan generar una nueva y pasando los parámetro requeridos por el algoritmo genético, en este caso el número de ciudades y el número de pobladores en cada generación.

5.3.1.6 Diseño en VHDL.

La implementación VHDL propuesta para el agente maestro representando las creencias y las intenciones consta de:

- Un generador de números pseudo aleatorio[41] implementando el algoritmo de Green (**figura 19**), el cual es un algoritmo rápido y cumple con los estándares FIPS 140-1 y FIPS 140-2 (*Federal Information Processing Standard*)[19], lo cual lo hace apto para uso en criptografía.

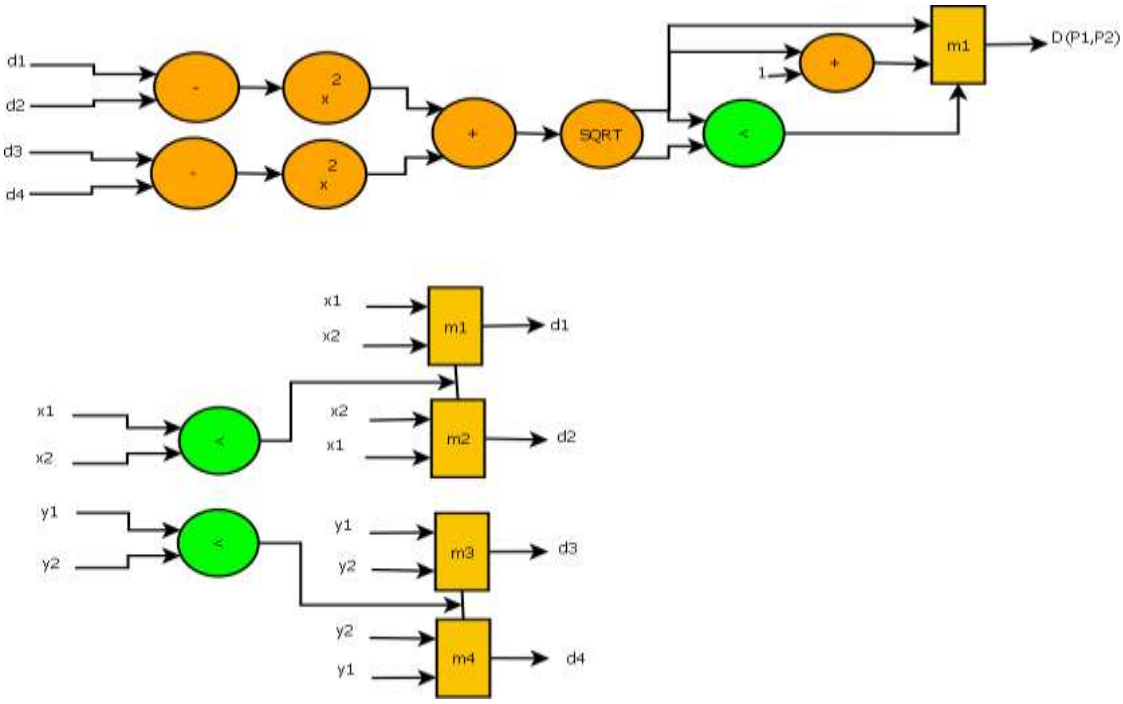
Figura 19 Implementación hardware algoritmo de *GREEN*.



[41] Ver ANEXO F

- un circuito para encontrar la distancia euclidiana entera entre dos puntos con redondeo del primer decimal (**figura 20**), el cual realiza en paralelo las operaciones aritméticas de resta, potenciación y suma, obteniéndose el resultado de la operación en un ciclo de reloj y una serie de registros, contadores, comparadores y multiplexores necesarios para realizar los diferentes movimientos de memoria.

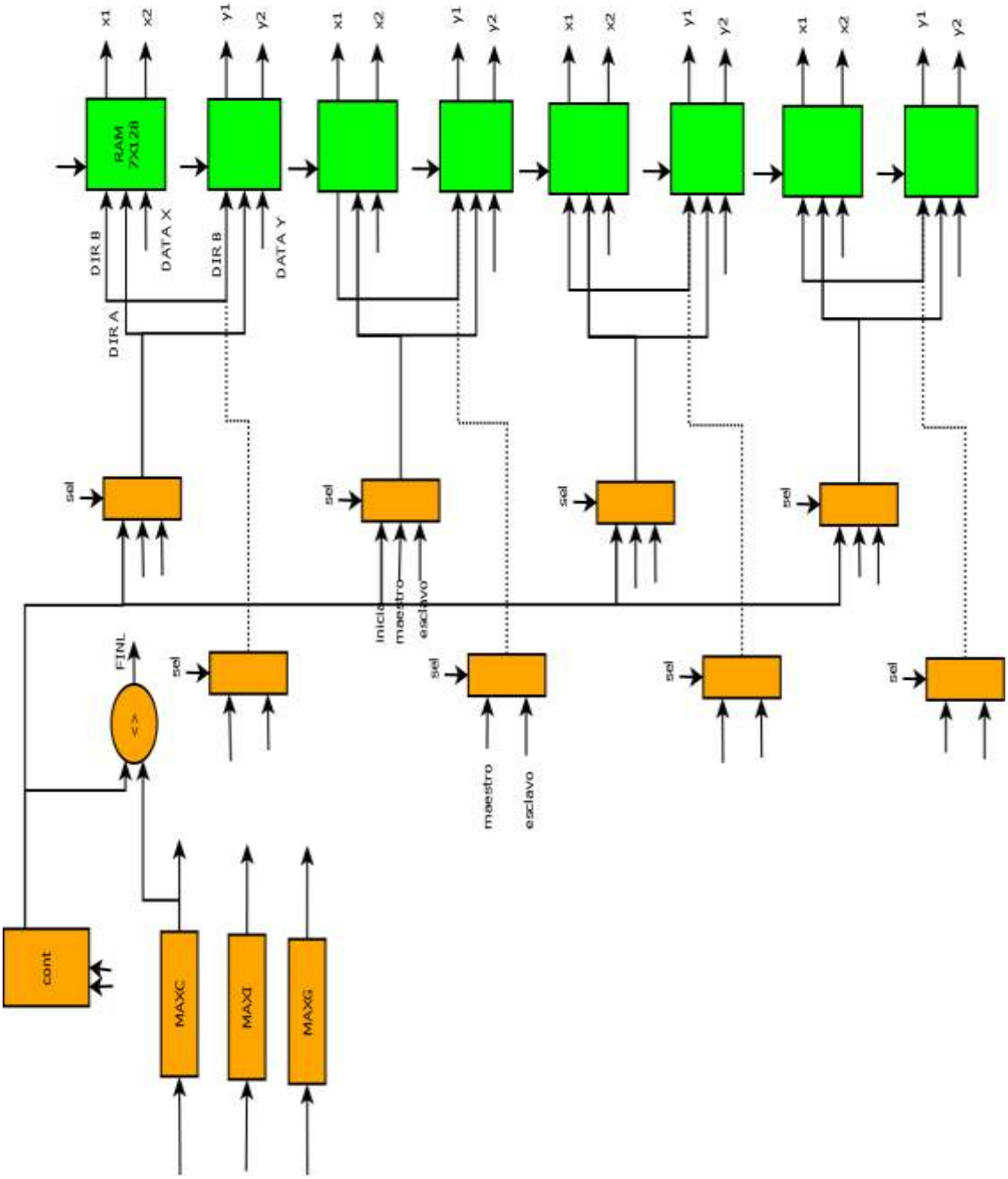
Figura 20 Implementación hardware distancia euclidiana.



- Un modulo que se encarga de cargar los datos del problema en memoria RAM (**Figura 21**), de forma independiente para cada modulo interno del agente maestro y para cada agente esclavo que haga parte del SMA. Este modulo comparte la información entre el agente maestro y los agentes esclavo según las operaciones que se estén realizando, si el agente maestro se encuentra en el estado de inicialización o búsqueda de los valores de adaptación de las poblaciones pertenecientes a cada modulo, el

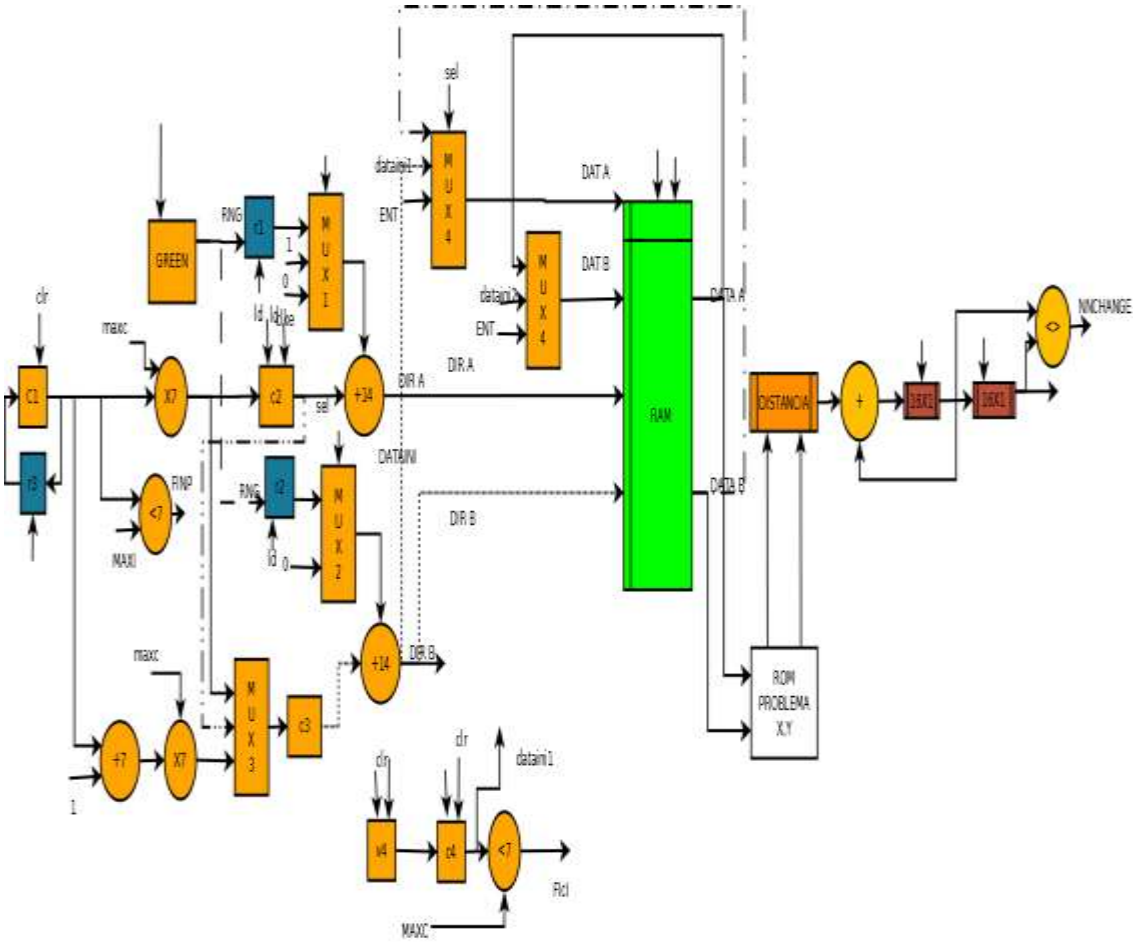
control de la memoria del problema esta en manos del agente maestro, de lo contrario, cede el control de acceso a la memoria a los agentes esclavo.

Figura 21 Arreglo de la memoria del problema.



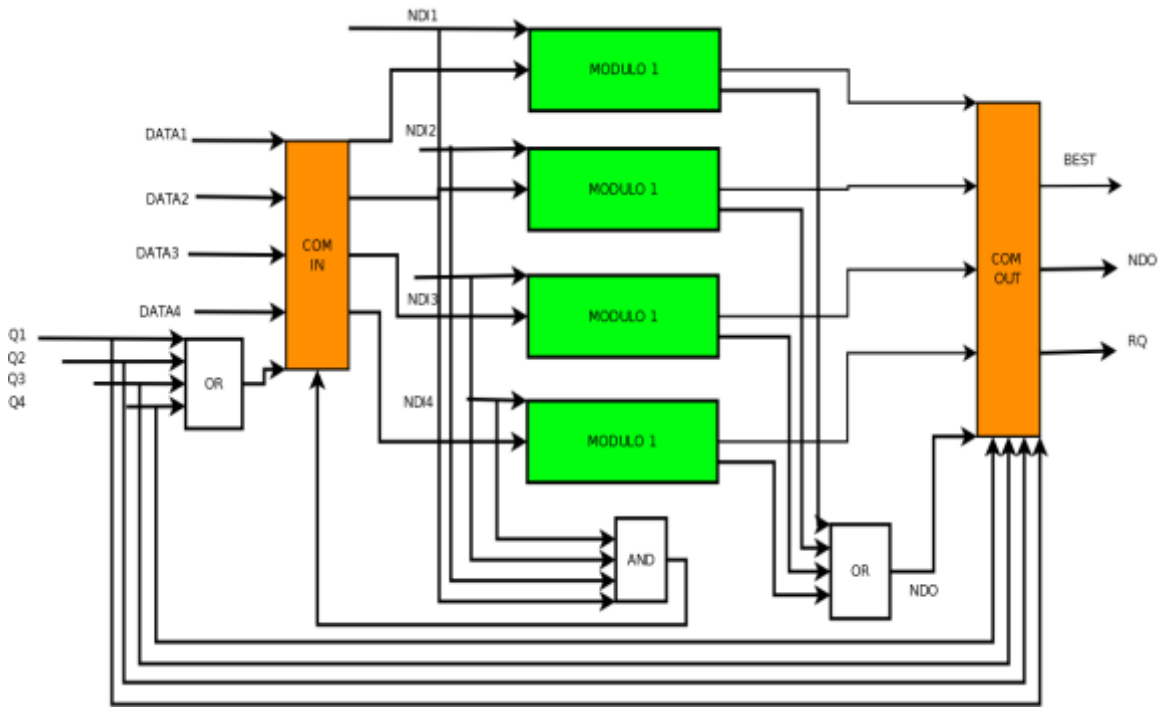
El agente maestro[42] está compuesto por 4 módulos iguales (**figura 22**), con los que se gana velocidad de procesamiento, ya que maneja de forma independiente la comunicación con los esclavos al igual que la inicialización y búsqueda de la solución. Adicionalmente por fuera de estos 4 módulos posee la circuitería necesaria para comunicación, determinación de presencia de esclavos y multiplexación de las salidas de los módulos (**figura 23**).

Figura 22 Módulo interno básico del agente maestro.



[42] Ver ANEXO G

Figura 23 Esquema básico agente maestro.



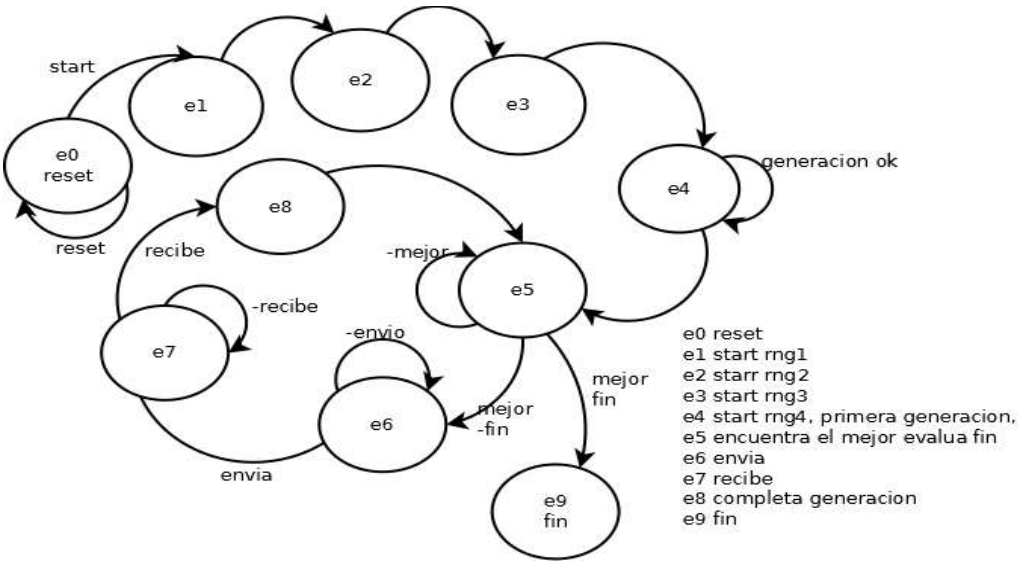
Las intenciones del agente maestro están determinadas por las máquinas de estado que mueven el circuito de forma congruente a la solución del problema, en ese sentido, tal y como los agentes software manejan una serie de estados mentales, en este diseño se siguió ese mismo lineamiento, teniendo una máquina de estados que determina el estado mental del agente y define cual es plan a seguir y el comportamiento actual del agente.

Donde el estado e_0 es el estado de *reset* en el cual el agente está inactivo, de los estados e_1 a e_4 inicializa los cuatro generadores de números pseudo aleatorios, teniendo en cuenta que hay uno por cada módulo haciendo que todos tengan diferentes semillas y por lo tanto secuencias diferentes. El estado e_4 es donde el agente carga los datos del problema, crea la primera población del algoritmo genético y busca que haya agentes esclavo, conectados al sistema o en funcionamiento.

En el estado e_5 se encuentra la mejor población, dada por aquella que posea el individuo mejor adaptado y verifica si ha terminado la tarea, en cuyo caso, da una respuesta al usuario en el estado e_9, en el estado e_6 la mejor población es enviada a los agentes esclavo, en el estado e_7 el agente maestro espera por las respuestas de los agentes esclavo, en el estado e_8 termina una generación, pasando a encontrar de nuevo el mejor de los individuos en el estado e_5, así hasta que el agente termine su tarea y dé una respuesta válida. Se debe tener en cuenta que para que el sistema funcione debe existir al menos un agente esclavo activo (**figura 24**).

Las transiciones de estado en esta FSM son causados por el fin de las tareas que se realizan en cada uno de estos.

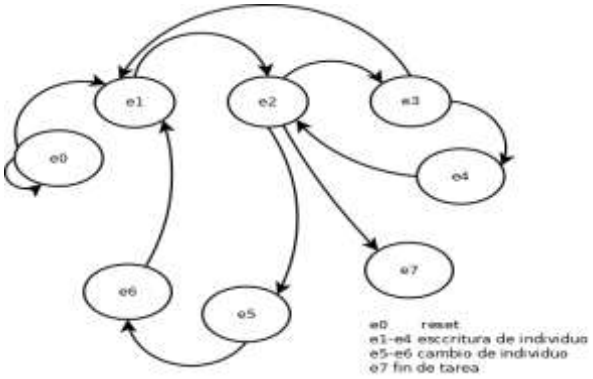
Figura 24 Representación mental del agente maestro com una FSM.



Internamente los objetivos de diseño o planes del agente son llevados a cabo por una máquina de estados que responde a los estados mentales externos del agente de la siguiente manera:

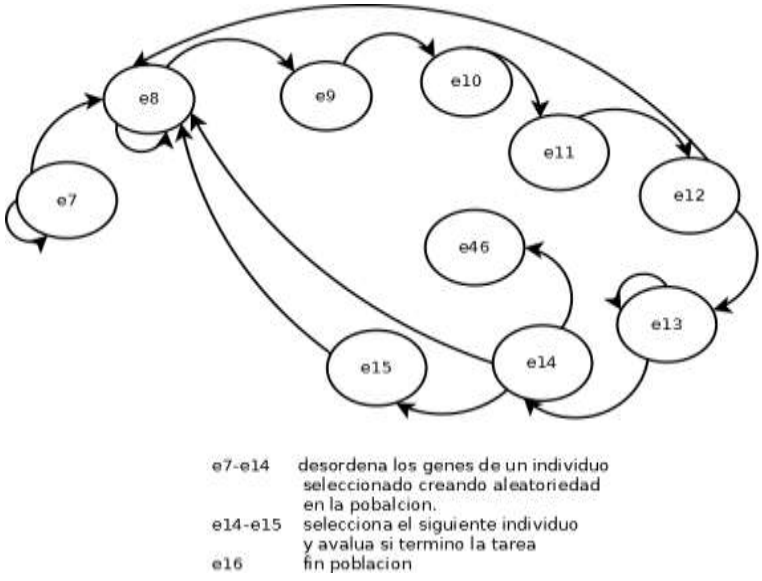
- Inicialización de la memoria RAM (**figura 25**), con el número de individuos requeridos por el usuario, cada uno con el número de ciudades dados por el problema, (e_0 a e_5).

Figura 25 FSM inicialización de la memoria RAM.



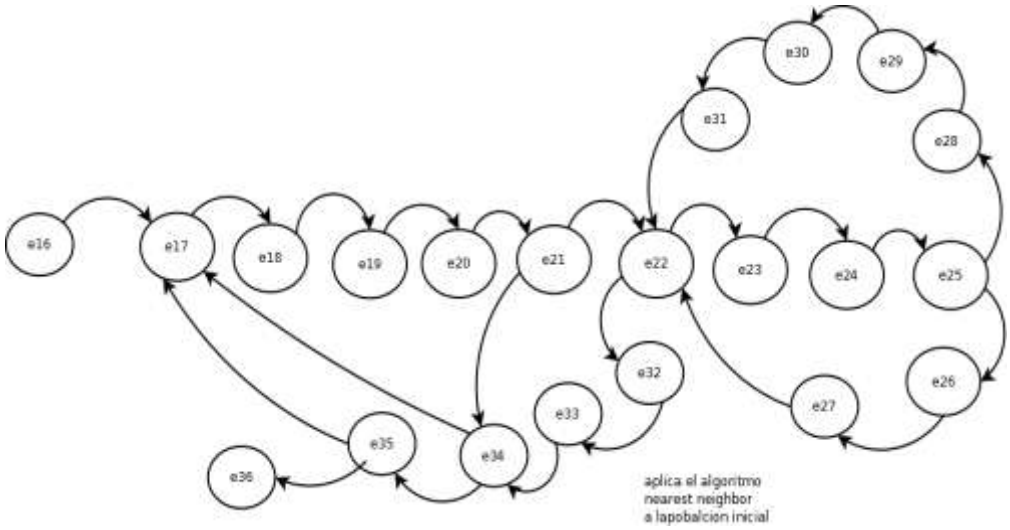
- Generar población aleatoria (**figura 26**), a partir de los individuos puestos en la memoria RAM (e_5 a e_{13}).

Figura 26 FSM ramdomización de la población.



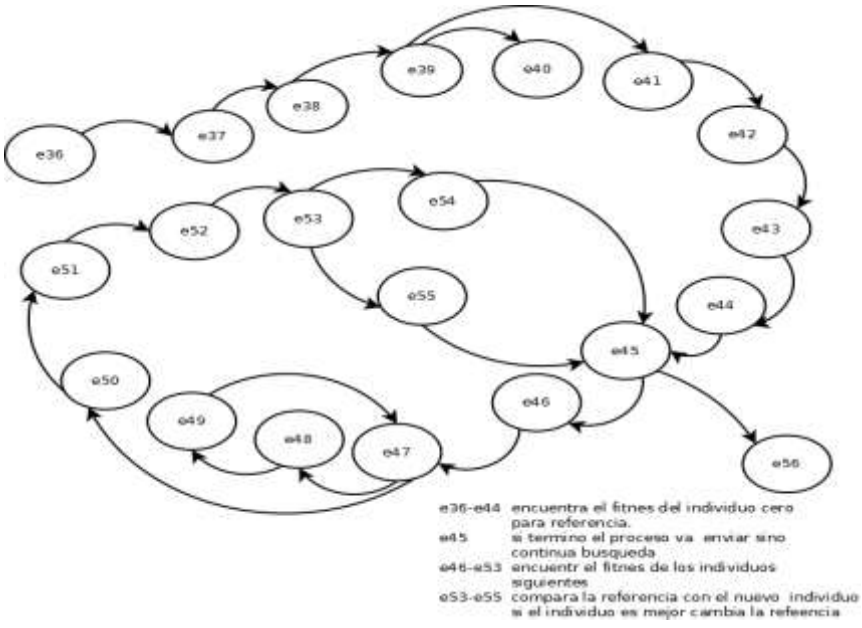
- Limitar el espacio de búsqueda aplicando el algoritmo del vecino más cercano (**figura 27**) a toda la población, (e_{14} a e_{26}).

Figura 27 FSM algoritmo vecino más cercano.



- Encuentra en las poblaciones generadas el individuo mejor adaptado (**figura 28**), es decir el que determine la ruta más corta hasta el momento (e_{36} a e_{55}).

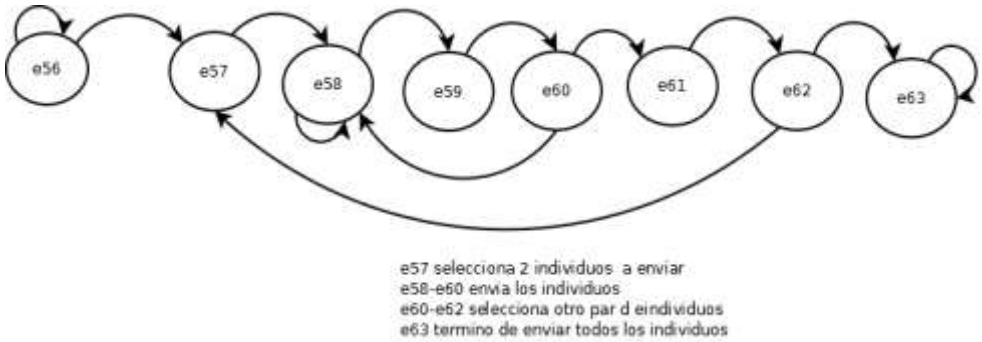
Figura 28 FSM encontrar mejor población.



- Envía la mejor población a los agentes esclavo activos y dispuestos a colaborar en la solución del sistema (**figura 29**), (e_{56} a e_{63}), en el estado e_{36}

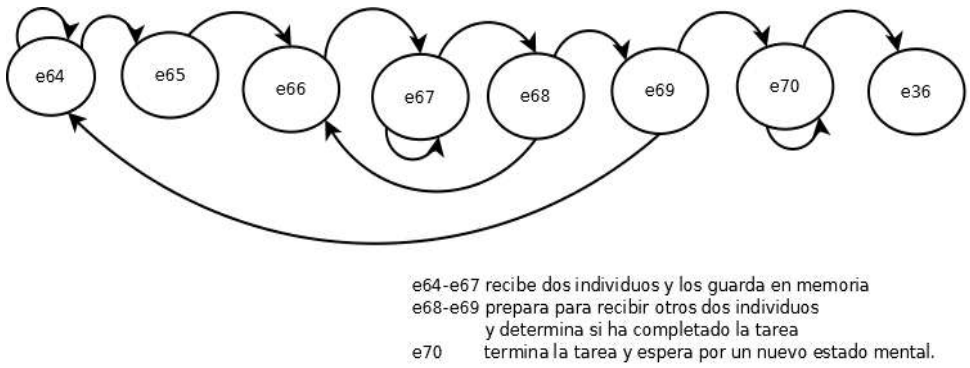
también se define si se continua con una nueva generación o se presenta una respuesta al usuario.

Figura 29 FSM envío de población.



- Recibe las nuevas generaciones de individuos enviadas por los agentes esclavo (**figura 30**), (e_{64}, a e_{70}), en el estado e_{70} regresa al estado de evaluación, para determinar la adaptación de las poblaciones recibidas y evaluar si termina la terea o continúa con la generación siguiente.

Figura 30 FSM recepción de individuos.



Al tener esta configuración es fácil cambiar los estados mentales haciendo que el agente tenga comportamientos diferentes, por ejemplo si se desea que el agente maestro no aplique el algoritmo del vecino más cercano a la población sino que en lugar de ello utilice una estándar totalmente aleatoria y pueda recorrer todo el espacio de búsqueda, solo se requiere cambiar el estado mental y no modificar los

estado internos o planes del agente, en este sentido solo se modifica el orden en que éstos son aplicados.

5.3.2 Agente esclavo hardware.

Este agente es el que se encarga de realizar las operaciones genéticas de selección cruce y mutación, es decir de encontrar una nueva población de individuos a partir de una existente, en este caso ésta es proporcionada por el agente maestro, este agente solo tiene interacción con el agente maestro y no con el usuario, los datos del problema y restricciones del mismo son proporcionados también por el agente maestro.

Al igual que el agente maestro las creencias del agente esclavo están determinadas por el problema a solucionar.

El agente esclavo está en capacidad de:

- Inicializar las semillas de los generadores pseudo aleatorios tal que todos tengan secuencias diferentes.
- Realizar los procesos de selección, cruce y mutación.
- Encontrar una nueva generación de individuos.
- Comunicarse con un agente maestro para la recepción y envió de la población.
- Informar al agente maestro si se encuentra o no disponible, tiene conocimiento de los individuos.
- Seleccionar de la población los mejores individuos para reproducción y eventualmente descartar los peores.

5.3.2.1 Modelo de creencias.

Las creencias del agente esclavo se limitan a una única población de individuos, el conocimiento del problema (posiciones de las ciudades , número de ciudades y

número de individuos encontrado en la población), de los mejores individuos dentro de dicha población, los padres seleccionados, los hijos de estos padres, y las probabilidades de cruce y de mutación de los individuos, algunos de stos datos son proporcionados por el agente maestro y otros son generados al interior del propio agente esclavo como es el caso de las probabilidades, padres e hijos.

5.3.2.2 Modelo de intenciones.

En el modelo de intenciones se tienen que el agente está en capacidad de seleccionar aleatoriamente padres aptos para reproducción implementando la selección por torneo, tomar un punto de cruce aleatorio y realizar las operaciones de seleccionar, copiar y pegar vectores, encontrar la distancia euclidiana entre dos puntos, comparar individuos cruce y/o mutación si los individuos seleccionados cumplen con estas probabilidades.

5.3.2.3 Modelo de deseos.

Haciendo uso de las intenciones, el agente está en capacidad de comunicarse de forma eficiente con un agente maestro, determinar la presencia y/o ausencia del agente maestro, ajustar el tamaño de memoria requerido para realizar las operaciones, identificar individuos independientes dentro de la población, encontrar las bondades de éstos y determinar cuáles son más aptos para reproducción. Los deseos del agente son los de generar una nueva generación a partir de una existente teniendo en cuenta la función de adaptación de los individuos.

5.3.2.4 Modelo del agente.

El agente esclavo está compuesto por un único módulo con capacidad para realizar todos los movimientos de memoria requeridos para realizar las operaciones genéticas (selección, cruce y mutación), además de poder comunicarse de forma eficiente con un agente maestro, a diferencia del agente

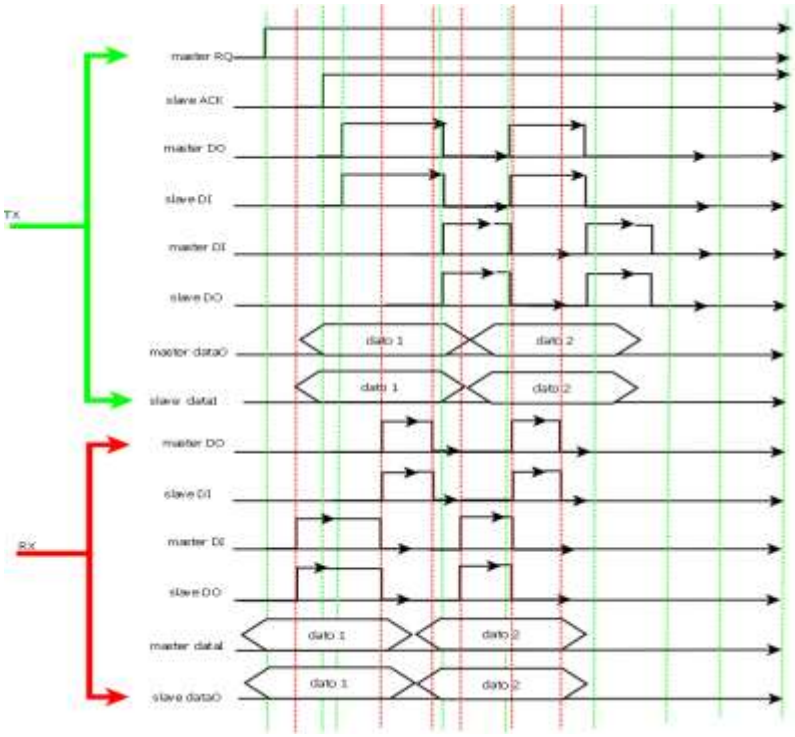
maestro que puede comunicarse con varis esclavos al tiempo, el agente esclavo solo obedece a un agente maestro.

5.3.2.5 Modelo de interacciones.

La comunicación se realiza teniendo en cuenta la presencia de una agente maestro y si éste solicita o no sus servicios.

Al recibir una petición del agente maestro, el agente esclavo responde si está o no disponible, acto seguido se prepara para la recepción de la población proveniente del maestro, la cual llega en forma serial recibiendo dos individuos al tiempo y guardándolos en las posiciones de memoria correspondientes. De la misma forma cuando se obtiene una nueva población, el agente esclavo envía esta población al agente maestro una vez éste acepte la transmisión enviándose en el mismo formato de dos individuos a la vez (**Figura 31**).

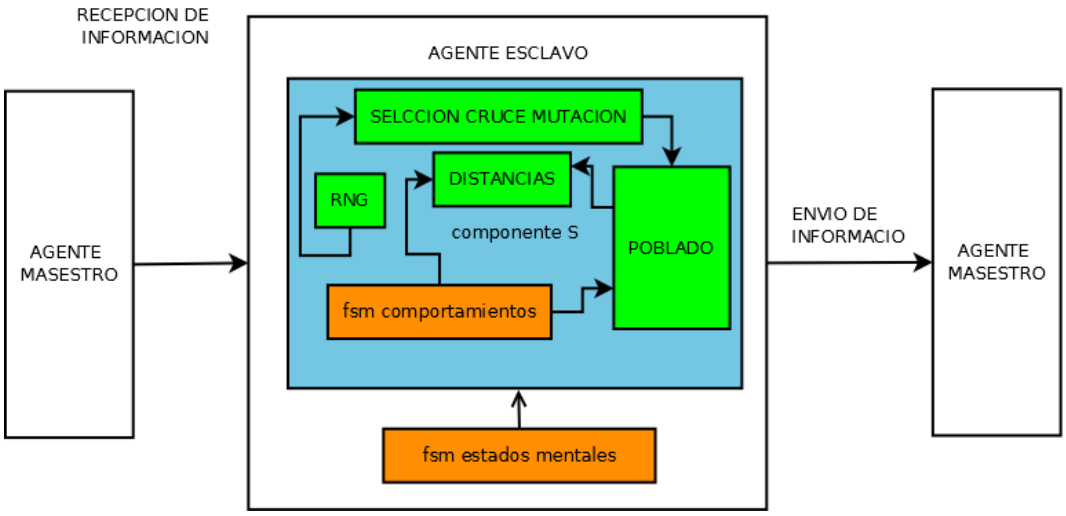
Figura 31 Comunicación entre agentes.



5.3.2.6 Diseño en VHDL.

En la implementación VHDL propuesta[43], el circuito de cada agente esclavo (**figura 32**), está compuesto por un único módulo que contiene a única población sobre la cual se efectúan todas las operaciones, Internamente este módulo posee una unidad de inicialización compuesta por cuatro generadores de números psuedo aleatorios para inicializar los porcentajes, selección de padres, puntos de cruce y puntos de mutación, requeridos en cada iteración para la creación de una nueva generación.

Figura 32 Esquema general del agente esclavo.



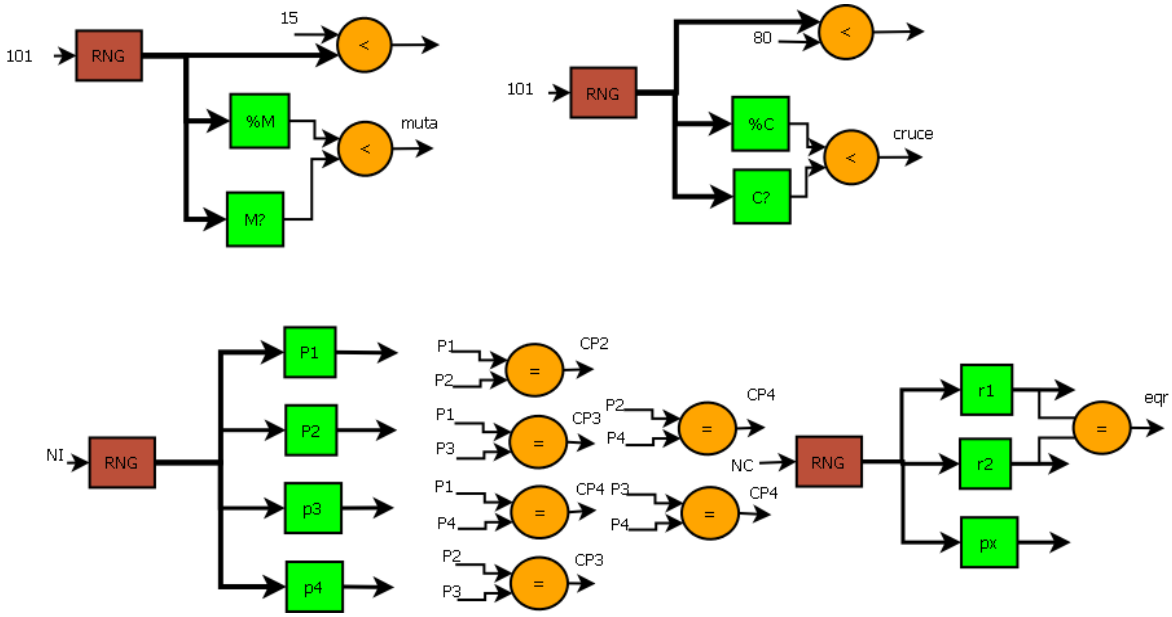
El generador de números pseudo aleatorios (RNG), y el modulo para determinar la distancia entre dos puntos es el mismo que en el agente maestro.

La forma en que se inicializan las variables de porcentaje de cruce, porcentaje de mutación, los padres que participan en el torneo (cuatro padres), el punto de cruce, y los puntos de mutación, son de forma aleatoria así que se implementó un arreglo de generadores pseudo aleatorios para agilizar la obtención de estos datos en cada ciclo, trabajando de forma paralela como se muestra en la **figura 33**. Y

[43] Ver ANEXO H

del mismo modo se evita que durante el proceso de selección sean escogidos padres iguales, asegurando que la población siempre esté cambiando y no se reemplacen individuos importantes en el proceso.

Figura 33 Arreglo de RNG´s para inicialización de los parámetros del algoritmo genético.



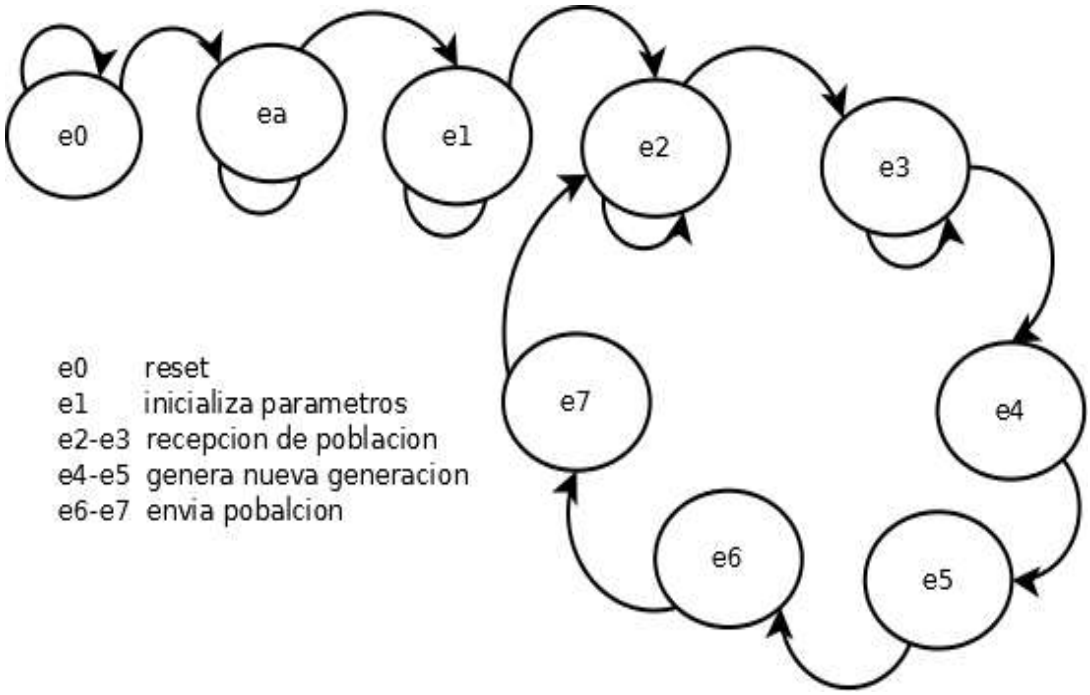
Los comportamientos del agente esclavo son diferentes a los del agente maestro por lo tanto el circuito requerido para realizar las operaciones también lo es de forma significativa, pues los movimientos en memoria requeridos para realizar las operaciones genéticas son bastante complejos.

Los estados mentales del agente esclavo (**figura 34**) están dados por la inicialización (estado e_1), la espera de una población para crear una nueva generación, (estados e_2-e_3), las operaciones de cruce mutación y selección, (estado e_4-e_5), y el envío de la nueva generación al agente maestro (estados e_6-e_7), que solicitó sus servicios.

Las propiedades que hacen del circuito propuesto un agente, se observan en la capacidad que tienen el circuito de tomar sus propias decisiones, de negar o aceptar un servicio requerido, de brindar siempre información verídica, y tomar las decisiones adecuadas para la modificación de sus creencias (población, padres , hijos), de la mejor forma posible según sus planes[44].

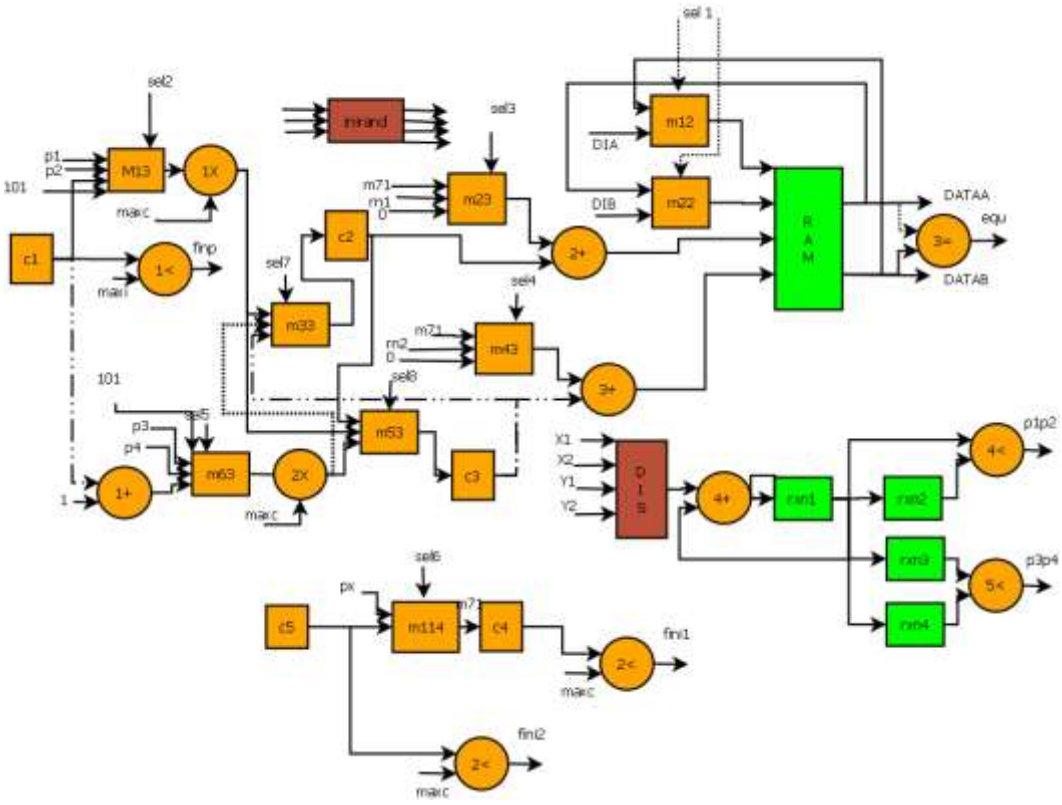
Los planes de agente esclavo están dados en una máquina de estados interna, la cual haciendo uso de las intenciones ejecuta los deseos modificando así sus creencias en cada estado.

Figura 34 Representación del estado mental del agente esclavo en una FSM.



[44] Estas son las propiedades que permiten definir el sistema dentro del paradigma de agentes.

Figura 35 Módulo componente del agente esclavo.



Los deseos del agente están representados de la siguiente manera:

- Inicializa las variables de P_c, P_m, P_1, P_2, P_3, P_4, C, M, M_1, M_2, P_x, que representan los porcentajes de cruce y mutación, los padres seleccionados para competir en torneo para la reproducción, los puntos de comparación para saber si hay o no cruce y/o mutación, los puntos de mutación y el punto de cruce. Esto es realizado en el módulo interno de inicialización, (estados e0-e1).
- Realiza la recepción de la población proveniente del agente maestro, (estados e2-e9).
- Realiza el torneo de los padres seleccionados, definiendo los padres que se van a reproducir y los que pueden ser reemplazados por un nuevo individuo producto de esta reproducción, (estados e9-e40).

- De acuerdo a las bondades de los padres y de las probabilidades de cruce y mutación (creencias), asignadas a estos que conforman las creencias del agente se realizan las operaciones según la **tabla 18**.

- El envió de los dato de regreso al agente maestro para su evaluación. (e148-e155).

Tabla 18 Operación que realiza el agente esclavo segun el estado de las creencias.

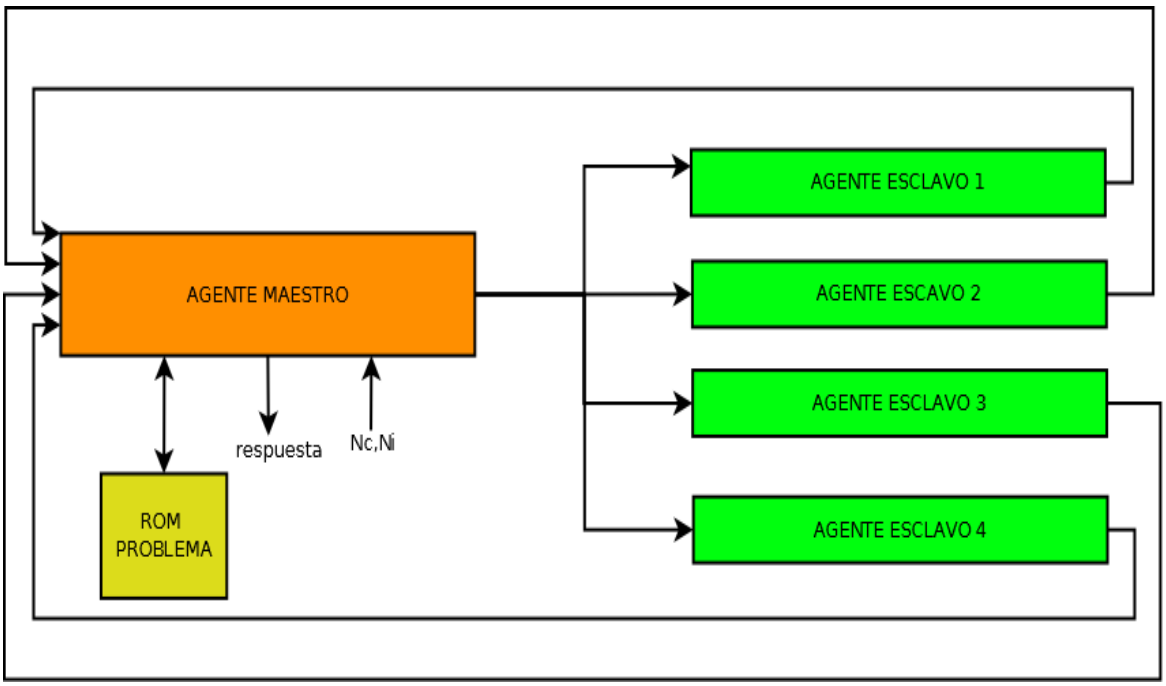
Operación	Reemplazo	Condición C	Condición M	estados
Mutación De P1	P3	C>PC	M<PM	e42-e46
Mutación De P2	P4	C>PC	M<PM	e61-e66
Mutación De P3	P2	C>PC	M<PM	ee55-e60
Mutación De P4	P1	C>PC	M<PM	e49-e54
Cruce De P2 Y P4	P3	C>PC	M<PM	e67-e83
Cruce De P1 Y P3	P4	C>PC	M<PM	e84-e99
Cruce De P3y P2	P1	C>PC	M<PM	e100-e115
Cruce De P4 Y P1	P2	C>PC	M<PM	e116-e131

5.3.3 Sistema multiagente hardware.

Un agente no puede resolver el problema presentado de forma individual, los agentes, maestro y esclavo fueron concebidos para trabajar de forma cooperativa para solucionar el problema del agente viajero y por lo tanto cada uno tiene una responsabilidad dentro del sistema. El trabajo en equipo de estos agentes implementan un algoritmo genético como instrumento motor para la solución del problema, así que el sistema completo está compuesto como mínimo por un agente maestro y un agente esclavo y por restricciones en el diseño, máximo por un agente maestro y hasta cuatro agentes esclavo.

En la **figura 36** se muestra una vista general de la configuración del sistema multiagente

Figura 36 Esquema general del sistema multiagente.



Obsérvese que la comunicación del maestro hacia los esclavos se hace a través de un mismo canal, es decir los esclavos reciben la misma información al mismo tiempo, pero la comunicación de los agentes esclavo hacia el agente maestro se hace de forma independiente.

5.4 PRUEBAS Y RESULTADOS.

Pruebas realizada en hardware (al sistetizar la descripción VHDL)[45] y software bajo las mismas condiciones del problema: cuatro agentes esclavo, un agente maestro, nuemero de generaciones igual al tamño del problema y número de pobladores igual a dos veces el tamaño del problema, donde se varía el número de ciudadesl (**tabla 19**). Los datos de las ciudades se toman de los problemas oliv30 eil51 y eil76.

[45] Ver ANEXO I

Tabla 19 Prueba del sistema multiagente hardware.

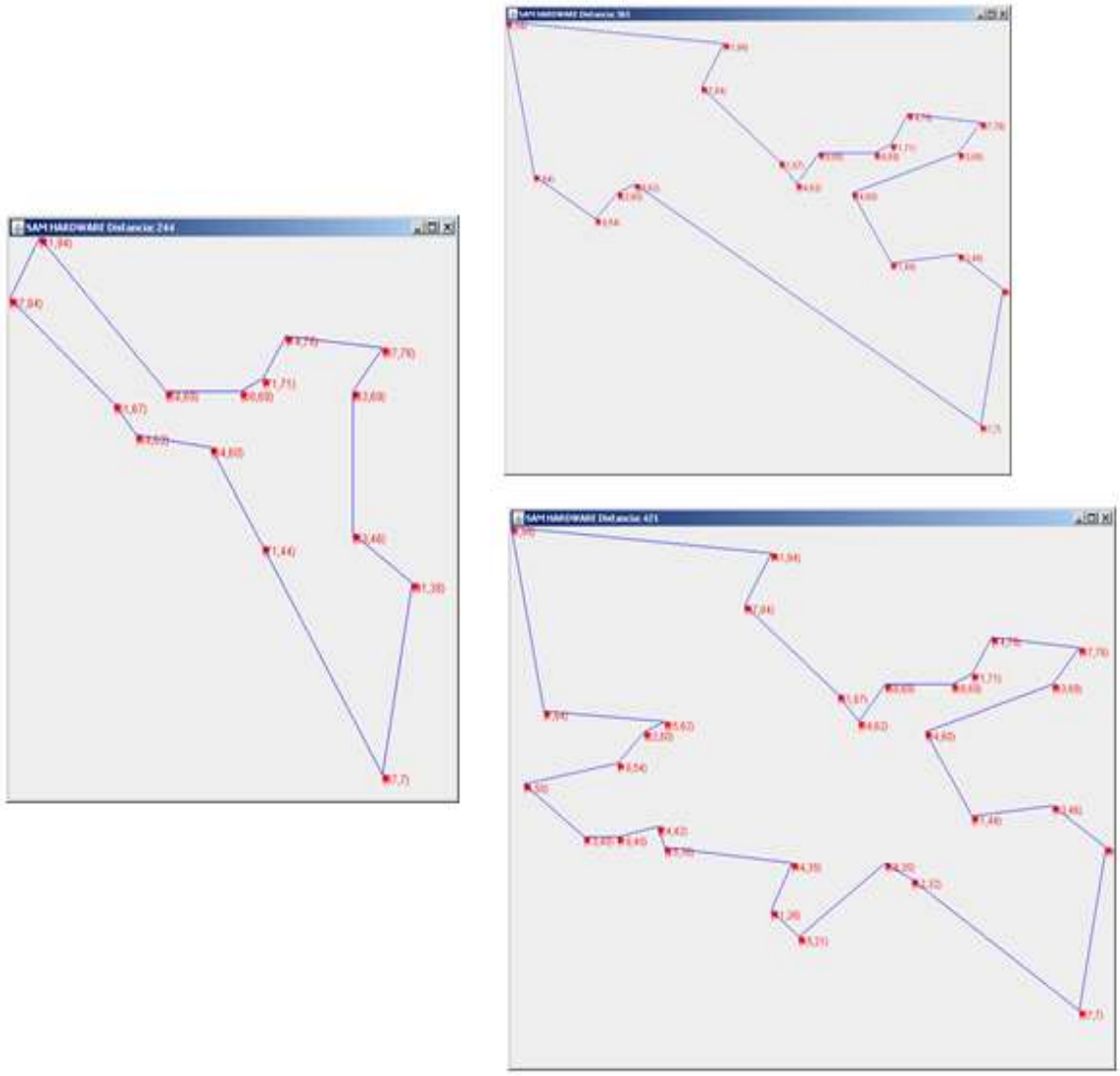
N	R	D	T
15	7,8,9,10,6,11,12,13,15,14,5,4,1,2,3	255	3.8
	10,6,11,15,14,13,12,5,4,1,2,3,7,8,9	244	3.2
	6,11,12,13,15,14,5,4,1,2,3,7,8,9,10	255	3.2
	2,3,7,8,9,10,6,11,15,14,13,12,5,4,1	244	3.2
	6,11,15,14,13,12,5,4,1,2,3,7,8,9,10	244	3.2
	PROMEDIO	248.8	3.3
20	12,11,6,10,9,8,7,5,4,3,2,1,19,18,20,17,16,15,14,13	365	8.2
	18,19,1,2,3,4,5,7,8,9,10,6,11,12,13,14,15,16,17,20	365	7.4
	1,2,3,4,5,7,8,9,10,6,11,12,13,14,15,16,17,20,19,18	365	7.4
	20,17,16,18,19,1,2,3,4,5,7,8,9,10,6,11,12,13,14,15	365	7.4
	18,19,1,2,3,4,5,7,8,9,10,6,11,12,13,14,15,16,17,20	365	7.4
	PROMEDIO	365	7.6
30	11,12,13,14,15,16,19,18,20,17,21,22,23,24,25,28,26,27,29,30,4,3,2,1,5,10,9,8,7,6	450	29
	19,17,16,14,15,13,12,11,6,10,9,7,8,5,4,3,2,1,30,29,27,26,28,24,25,23,22,21,20,18	437	29
	20,21,22,23,24,25,28,26,27,29,30,1,2,3,4,5,12,13,11,6,10,8,7,9,15,14,16,17,19,18	432	27
	20,18,19,17,16,15,14,13,12,11,6,10,9,8,7,5,4,3,2,1,30,29,27,26,28,25,24,23,22,21	421	28
	19,17,16,15,14,13,12,11,6,10,9,8,7,5,4,3,2,1,30,29,27,26,28,24,25,23,22,21,20,18	424	29
	PROMEDIO	432.8	28.4

PROBLEMA	HARDWARE			SOFTWARE*		
N	D	T	E	D	T	E
15	248.8	3.3	1.9%	256.8	666.6	4.8%
20	365	7.6	6.1%	365	980.4	6.1%
30	432.8	28.4	2.7%	446.6	2526	5.7%

Tras comparar los resultados de las implementaciones hardware y software, se encuentra que los resultados en cuanto a la calidad de la solución son bastante buenos en ambas implementaciones, sin embargo el tiempo invertido en la implementación hardware es sustancialmente menor y la media de las soluciones para el mismo problema en varias ejecuciones, también lo es, estando las soluciones del SMA hardware hasta un 3% mejor situadas que las del SMA

software con respecto de la solución optima. Y logrando mejoras porcentuales de velocidad hasta en un 20000%

Figura 37 Mejores soluciones obtenidas por el SMA hardware para problemas de 15, 20 y 30 ciudades.[46]



[46] Graficas obtenidas con una herrameienta JAVA para graficar los datos obtenidos en simulación. Ver ANEXO J

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO.

- A pesar del acelerado desarrollo de los sistemas informáticos en los últimos años y debido a que el paradigma de agentes es relativamente muy nuevo aún hace falta mucho camino por recorrer en cuanto a metodologías de diseño, lenguajes de programación que soporten el paradigma y herramientas de desarrollo más especializadas.
- Las plataformas existentes están aún en desarrollo y esto hace difícil encontrar documentación y soporte, muchos de los proyectos que se encuentran al respecto han sido abandonados.
- Los lenguajes orientados a objetos como java y C# aunque se asemeje mucho al paradigma de agentes no lo enmarca completamente dejando muchos de sus atributos por fuera, pero los lenguajes orientados a agentes aún están en las primeras etapas de desarrollo por ese hecho los diseñadores han optado por el uso de frameworks como jade y jadex.
- La plataforma jade brinda grandes beneficios ya que hace una abstracción del concepto de agente y libera al programador para que se enfoque más en la solución del problema.
- La implementación de sistemas de agentes sobre hardware reconfigurable FPGAs es posible, además de poder abstraer las metodologías de desarrollo

software hacia un entorno hardware con algunas modificaciones e interpretaciones conceptuales.

- Al observar los recursos requeridos para que un sistema multiagente funcione sobre la plataforma JADE, para el caso, resolviendo el problema del agente viajero, se determina que la exigencia en procesamiento y memoria es considerable, llevando una vez más a la idea de que la implementación de sistemas de agentes sobre hardware es ideal para problemas que requieran un alto procesamiento y demandas de recursos del sistema.
- El concepto de sistemas de agentes puede ser implementado en el área de bio-simulación a la hora de realizar simulaciones de sistemas biológicos complejos en los que se requiera independencia en sus partes constitutivas.
- En las pruebas efectuadas se observa que bajo los mismos parámetros, el sistema multiagente hardware es hasta 100 veces más rápido que la implementación software.

Para trabajos futuros se propone la implementación de un sistema multiagente hibrido software-hardware donde el grueso del procesamiento de ejecute en el hardware y en le software se encuentren los agentes de interfaz. Implementar este sistema hiblrdo puede mejorar considerablemente los procesos de simulacion, por ejemplo de sistemas biológicos, sin dejar de lado la visualización e interaccion con el usuario, implementando sistemas multiagente sobre FPGA en las áreas de bio-simulación, facilitando la integración de subsistemas aprovechando las características de comunicación de los agentes y sus demás propiedades.

REFERENCIAS BIBLIOGRÁFICAS.

[1] M. Ana, *Agentes Software Y Sistemas Multi-agente, Conceptos Arquitecturas Y Aplicaciones*. 2005.

[2] M. Wooldridge, C. Street, M. Manchester, and N. R. Jennings, “Intelligent Agents : Theory and Practice,” no. January, pp. 1-62, 1995.

[3] P. D. Valencia, “Agentes Inteligentes : el siguiente paso en la Inteligencia Artificial,” pp. 95-99, 2000.

[4] A. S. Rao and M. P. George, “BDI Agents : From Theory to Practice.”

[5] A. G. M. G. Ga, “Arquitectura Tolerante A Fallos Mediante Un Sistema Multiagente Para El Sistema De Control De Un Robot Movil,” 2007.

[6] G. D. F. Jose, L. L. Faraon, and R. A. Ramon, “Breve análisis de algunas metodologías de diseño de SMA,” 2004.

[7] P. A. S. Yanis, “Aplicacion De Metodologias Igenias, Zeus, Masina, Al Desarrollo De Sistemas Multiagente, Partiendo De Sma De Subastas Para La Identificacion De Mejores Practicas,” 2007.

[8] “AGENTES Y JADE Introducción : FIPA (Foundation for intelligent Physical Agents):”

[9] L. A. Monteserin, “JADE – Java Agent DEvelopment Framework Taller de sistemas multiagentes Introducción Características Plataforma multiagente.”

[10] B. Diaz Agudo and R. Fuentes Fernández, “Introducción a Jade.” pp. 1-15, 2003.

[11] I. Systems, U. Teknologi, and W. Virginia, “Engineering JADEX Agents with the MaSE Methodology Faculty of Computer Science and Computer Science and Electrical.”

[12] V. J. Adrian, “Algoritmos genéticos 2006-2007,” 2007.

[13] Y. Elhaddad and O. Sallabi, “A New Hybrid Genetic and Simulated Annealing Algorithm to Solve the Traveling Salesman Problem,” vol. I, pp. 0-3, 2010.

[14] M. Golub, L. Budin, and H.- Zagreb, “An Asynchronous Model of Global Parallel Genetic Algorithms.”

[15] I. Skliarova, “FPGA-based Implementation of Genetic Algorithm for the Traveling Salesman Problem and its Industrial,” 2001.

[16] H. R. Naji, L. Etzkorn, and B. E. Wells, “Applying multi agent techniques to reconfigurable systems,” *Advances in Engineering Software*, vol. 35, no. 7, pp. 401-413, Jul. 2004.

[17] Y. Meng, “Agent-based reconfigurable architecture for real-time object tracking,” *Journal of Real-Time Image Processing*, vol. 4, no. 4, pp. 339-351, Mar. 2009.

[18] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, “Flexible Implementation of Genetic Algorithms on FPGAs.”

[19] A. M[a] and G. Sevillano, “Circuitos Digitales basados en FPGAs para Generación de Números Aleatorios,” pp. 1-9, 1999.

[20] http://www.ececs.uc.edu/~abaker/JAFMAS

[21] http://www.alphaworks.ibm.com/tech/able

[22] https://www.ohloh.net/p/zeusagent

[23] http://www.madkit.org/

[24] http://grasia.fdi.ucm.es/main/es

[25] http://www-e.uni-magdeburg.de/mertens/TSP/index.html

ANEXOS.

ANEXO A.

Seleccion de la plataforma y metodología software para la implementación del sistema .

Durante el desarrollo del sistema multiagente se debió escoger la plataforma (**tabla 20**), sobre la cual se trabajaría, asi mismo como la metodología. Teniendo en cuenta aspectos y propiedades de las diferentes plataformas y metodologías existentes se llegó a la conclusión de que la mejor plataforma es la plataforma JADE. Y para el caso de la metodología, la BDI (**tabla 21**).

Tabla 20 Seleccion de la plataforma software.

Prop / Plat	ED	CM	BDI	CG	DT	AO	DOC	AV	FC	MOV	FRI	ADD
MATLAB	SI	NO	NO	NO	SI	NO	SI	SI	NO	NO	NO	NO
JADE	SI	SI	SI	NO	SI	SI	SI	SI	SI	SI	SI	SI
JADEX	SI	SI	SI	NO	SI	SI	NO	SI	SI	SI	SI	NO
MAD	NO	SI	NO	SI	SI	SI	SI	NO	SI	NO	SI	NO
JAF	NO	SI	NO	SI	SI	SI	SI	NO	SI	NO	SI	NO
ZEUS	NO	SI	SI	SI	SI	SI	SI	SI	SI	NO	SI	NO
C#	SI	NO	NO	NO	NO	NO	SI	SI	NO	NO	NO	NO
C++	NO	NO	NO	NO	NO	NO	SI	SI	SI	NO	NO	NO

ED= en desarrollo **CM**= soporta comunicaciones **BDI**= soporta BDI **CG**= generación de código **DT**= depuración **AO**= orientado a agentes **DOC**= documentación disponible **AV**= disponibilidad **FC**= código libre **MOV**= movilidad **FRI**= independiente de la plataforma **NAK**= sin conocimiento adicional.

Tabla 21 Seleccion de la metodologia.

Arq. / Pro	AO	SWT	HWI	PL	OB	BL	DOC	HLP	SMA	HC
BDI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI
MASE	NO	SI	NO	NO	NO	NO	SI	SI	SI	NO
GAIA	SI	NO	NO	NO	NO	NO	SI	NO	SI	NO
MASCOMMONKADS	NO	NO	NO	NO	NO	NO	SI	SI	SI	NO
VOWEL	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
UML	NO	SI	NO	NO	NO	NO	SI	NO	NO	NO
MESSAGE	NO	NO	NO	NO	SI	NO	SI	NO	NO	NO

AO= orientado a agentes **SWT**= herramientas software **HWI**= hardware **PL**=planes **OB**=objetivos **DOC**= documentación disponible **HLP**= ayuda disponible **FC**= código libre **SMA**= soporte multiagente **HC**= características humanas.

ANEXO B.

Código JAVA agente maestro

```
package geneticAl;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.Agent;
import java.io.*;
import java.util.Random;
import javax.swing.JFrame;
import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
public class AgentMaster extends Agent {
public String target;
public String X;
public String Y;
public String Pcruce;
public String Pmutacion;
//listado de agentes esclavo
private AID [] AgentesS; //agentes esclavo
static int[][] mTsp, mPoblacion,Sol;//,poblado;
tatic int[] vectorS;
static int Nciudades;
static int Npobladores;
int pmut,pcruz;
int alea;
static int gen,k1,k2,k3;
int solcounter,lastcounter;
int generacion,prob=0,muta=0,cruz=0,Ngeneracion;
int[] distancias;
int [] fitnes;
int [] ruleta;
int padre1,padre2,dish1,dish2,Best,media=0,worse=0;
long totalT=0;
int[] padre = new int[Nciudades];
int[] madre = new int[Nciudades];
int[] hijo = new int[Nciudades];
int[] hija = new int[Nciudades];
int[] Mejor= new int[Nciudades];
long tinicio,tfinal;
```

```
protected void setup() {
System.out.println("Hello World! My name is "+getLocalName());
solcounter=0;
lastcounter=0;
Object[] args = getArguments();
target="\\Users\\paula\\workspace\\geneticAL\\src\\geneticAl\\eil4.tsp";
System.out.println("El problema  solucionar es "+target);
ArchiveLoader Lectura = new ArchiveLoader();
Lectura.LeerArchivo("C:"+target);
mTsp = Lectura.getCiudades();
Nciudades=Lectura.getNCiudades();
Npobladores=30;
//************************************
Ngeneracion=15;
generacion=0;
 addBehaviour(new StepBehaviour());
}
public class StepBehaviour extends Behaviour {
 private int step = 1;
private AID bestSeller; //
private int bestPrice;  //
private int repliesCnt = 0; //
private MessageTemplate mt; //
int[][] GenNext = new int[Npobladores][Nciudades];
public void action() {
switch (step) {
    case 1:// inicializa
          Distance Distancia=new Distance(Npobladores,Nciudades);
          CopyVector copiar= new CopyVector(Nciudades,Npobladores);
          Fitnes ajuste=new Fitnes(Npobladores,Nciudades);
          FirstGeneration Poblacion= new FirstGeneration(Npobladores,Nciudades);
          Random random = new Random();
          Nearest cercano=new Nearest(Nciudades);
          Poblacion.Generation();//primera generacion
          mPoblacion=Poblacion.obtenerPoblado();
          for(int h=0;h<Npobladores;h=h+1)
          {
           mPoblacion[h]=cercano.Cercano(mTsp, mPoblacion[h]);
          }
          Distancia.CalcularDistancia(mTsp, mPoblacion);
          distancias=Distancia.getDistance();
          ajuste.calculaFitnes(distancias);
          Best=ajuste.getBest();
          worse=ajuste.getWorse();
          Mejor=copiar.CopiarVector(mPoblacion[Best]);
          mPoblacion[worse]=copiar.CopiarVector(Mejor);
          Distancia.CalcularDistancia(mTsp, mPoblacion);
          Distancias=Distancia.getDistance();
          ajuste.calculaFitnes(distancias);
```

```
        fitnes=ajuste.getFit();
        lastcounter=distancias[Best];//captura la mejor distancia
        media=ajuste.getMedia();
        target="";
        X="";
        Y="";
        do
        {
         pcruz=Math.abs(random.nextInt()%(101));
        }
        while (pcruz<90);// prorcentaje de cruce entre el 70 % y el 100%
        pmut=Math.abs(random.nextInt()%(5));//porcentaje de mutacion menor al 30%
        for(int j=0;j<Nciudades;j++)
        {
        target=target+Integer.toString(Mejor[j]);
        X=X+Integer.toString(mTsp[j][0]);
        Y=Y+Integer.toString(mTsp[j][1]);
        if(j<Nciudades-1)
        {
        X=X+',';
        Y=Y+',';
        target=target+',';
        }
        }
        step=2;
        break;
case 2:// busca agentes esclavos
         DFAgentDescription template = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("msm-pga");//tipo de agente o servicio que busca
        template.addServices(sd);
        try {
        DFAgentDescription[] result = DFService.search(myAgent, template);
        if(result.length > 0)
                {
                System.out.println("se han encontrado los agentes esclavo:");
                AgentesS = new AID[result.length];
                for (int i = 0; i < result.length; ++i)
                 {
                   AgentesS[i] = result[i].getName();
                  System.out.println(AgentesS[i].getName());
                  }
         step=3;
                }
        Else
                {
                System.out.println("no se han encontrado agentes esclavo");
                AgentesS = new AID[result.length];
                }
```

```
        }
        catch (FIPAException fe) {
        System.out.println(" error de comunicacion");
        fe.printStackTrace();
        doDelete();
        }
         break;
 case 3:
        System.out.println("agente maestro ");
        // envia la propuesta a todos los agentes necontrados
        ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
        for (int i = 0; i < AgentesS.length; ++i)
        {
        cfp.addReceiver(AgentesS[i]);
        }
//en target queda almacenado el mejor individuo de la generacion 0
//en forma de string
        cfp.setContent(target);//envia el mejor individuo a cada agente
        cfp.setConversationId("TSP");
        cfp.setReplyWith("cfp-"+System.currentTimeMillis()); // vaolor unico
        myAgent.send(cfp);
        cfp.setContent(X);//envia ecoordenadas X
        cfp.setConversationId("TSP");
        cfp.setReplyWith("cfp-"+System.currentTimeMillis()); // vaolor unico
        myAgent.send(cfp);
        cfp.setContent(Y);//envia coordenadas Y
        cfp.setConversationId("TSP");
        cfp.setReplyWith("cfp-"+System.currentTimeMillis()); // vaolor unico
        myAgent.send(cfp);
        cfp.setContent(Integer.toString(Npobladores));//envia coordenadas Y
        cfp.setConversationId("TSP");
        cfp.setReplyWith("cfp-"+System.currentTimeMillis()); // vaolor unico
        myAgent.send(cfp);
        cfp.setContent(Integer.toString(Nciudades));//envia coordenadas Y
        cfp.setConversationId("TSP");
        cfp.setReplyWith("cfp-"+System.currentTimeMillis()); // vaolor unico
        myAgent.send(cfp);
        System.out.println("envia ciudades");
// prepara un template para las propuestas
// aqui se debe enviar la informacion adicional
//envia el vector de posiciones x,y
// se envia la probabilidad de mutacion diferente a cada agente
// se envia la probabilidad de cruce diferente para cada agente
// el resto de calculos se hacen en el agente escalvo
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("TSP"),
        MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
        step = 4;
        tinicio = System.currentTimeMillis();
        break;
```

```
case 4:// espera la respuesta----
        Distance Distancia1=new Distance(Npobladores,Nciudades);
        Fitnes ajuste1=new Fitnes(Npobladores,Nciudades);
        CopyVector copiar1= new CopyVector(Nciudades,Npobladores);
        Roulett Ruleta=new Roulett(Npobladores,Nciudades);
        ACLMessage reply = myAgent.receive(mt);
        if (reply != null)
         {
// respuesta recibida
         if (reply.getPerformative() == ACLMessage.CFP)
          {
// acepta y guarda la respuesta
// se convierte a enteros y conforma una nueva generacion
// de individuos
        target = reply.getContent();
        for (int k=0; k<Nciudades;k++)
         {
        Mejor[k]=Integer.parseInt(target.split(",")[k].trim());
        }
        mPoblacion[worse]=copiar1.CopiarVector(Mejor);
        Distancia1.CalcularDistancia(mTsp, mPoblacion);
        distancias=Distancia1.getDistance();
        ajuste1.calculaFitnes(distancias);
        worse=ajuste1.getWorse();
        repliesCnt++;
        if (repliesCnt >= AgentesS.length)
        {
// se reciben todas las respuestas
        Best=ajuste1.getBest();
        Mejor=copiar1.CopiarVector(mPoblacion[Best]);
        fitnes=ajuste1.getFit();
        Ruleta.Ruleta(fitnes);
        ruleta=Ruleta.getRoul();
        step = 5;
        repliesCnt=0;
        }
        }
        }
        else {
        block();
        }
        break;
case 5:
        PrintVector imprime2=new PrintVector(Npobladores,Nciudades);
        target="";
        for(int j=0;j<Nciudades;j++)
        {
        target=target+Integer.toString(Mejor[j]);
           if(j<Nciudades-1)
```

```
            {
           target=target+',';
             }
}
  generacion++;
 if(lastcounter>distancias[Best]) //si hay un cambio en la mejor solucion
{
solcounter=generacion;
lastcounter=distancias[Best];
}
 System.out.println("GENERACION:"+generacion+ "DISTANCIA"+
distancias[Best]); if(generacion<Ngeneracion)
{
ACLMessage cfp2 = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < AgentesS.length; ++i)
 {
cfp2.addReceiver(AgentesS[i]);
}
bestPrice=9;
cfp2.setContent(Integer.toString(bestPrice));
cfp2.setConversationId("TSP");
cfp2.setReplyWith("order "+System.currentTimeMillis());
myAgent.send(cfp2);
cfp2.setReplyWith("order "+System.currentTimeMillis());
cfp2.setContent(target);
myAgent.send(cfp2);
mt = MessageTemplate.and(MessageTemplate.MatchConversationId("TSP"),
MessageTemplate.MatchInReplyTo(cfp2.getReplyWith()));
step=4;
 }
 else
  {
  tfinal = System.currentTimeMillis();
 ACLMessage cfp2 = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < AgentesS.length; ++i)
{
cfp2.addReceiver(AgentesS[i]);
}
bestPrice=1;
cfp2.setContent(Integer.toString(bestPrice));
cfp2.setConversationId("TSP");
cfp2.setReplyWith("order "+System.currentTimeMillis());
myAgent.send(cfp2);
System.out.println("ruta");
imprime2.printV(Mejor,Nciudades);
System.out.println("distancia="+distancias[Best]);
System.out.println("tiempo="+(tfinal-tinicio));
System.out.println("generacion solucion :"+solcounter);
vectorS=mPoblacion[Best];
```

```
            Sol= new int[Nciudades][2];
            for(int k=0;k<Nciudades;k++)
            {
            Sol[k][0]=mTsp[vectorS[k]-1][0];
            Sol[k][1]=mTsp[vectorS[k]-1][1];
            }
            JFrame ventana = new JFrame();
            Lienzo lienzo = new Lienzo(Sol,Nciudades);
            ventana.add(lienzo);
            ventana.setSize(800,650);
            ventana.setTitle("Distancia:"+distancias[Best]+ " Tiempo:" +(tfinal-tinicio) + "
            mseg");
            ventana.setVisible(true);
            step = 6;
            }
            break;
case 6:
            break;
              }


}
public boolean done()
 {
 return step == 6;
}
public int onEnd() {
myAgent.doDelete();
return super.onEnd();
}   }    // END
}
```

ANEXO C.

Código JAVA agente esclavo

```
/*****************************************************************
Agente esclavo encargado de hacer las operaciones de mutacion y
cruce y obtencion de distancia de las posibles soluciones

 *****************************************************************/
package geneticAl;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.Agent;
import java.io.*;
import java.util.Random;
import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
public class AgenteSlave extends Agent {
public String target;
public String X;
public String Y;
public int salir;
public int daticos[][];//ubicacion de las ciudades en el plano (x,y)
public int ciudades;//numero de ciudades a visitar
public int contratado;//si el agente ya hasido contratado por otro    maestro
public int solucion;
static int[][] mTsp, mPoblacion,Sol,GenNext;//,poblado;
static int[] vectorS;
static int Nciudades;
static int Npobladores;
static int pmut,pcruz;
static int alea;
static int gen,k1,k2,k3;
int generacion,prob=0,muta=0,cruz=0;
int[] distancias;
int [] fitnes;
```

```
int [] ruleta;
int[] padre;
int[] madre;
int[] hijo ;
int[] hija ;
int[] Mejor;
int padre1,padre2,dish1,dish2,Best,media=0,worse=0;
long totalT=0;
private AID [] AgentesS;
MessageTemplate mt = essageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg;
ACLMessage reply;
protected void setup() {
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("msm-pga");
sd.setName("esclavo");
dfd.addServices(sd);
try
        {
        DFService.register(this, dfd);//se regitra en el directorio
        }
catch (FIPAException fe)
         {
        fe.printStackTrace();
        }
// Add the behaviour
addBehaviour(new respuestaAlMaestro());
 }
public class respuestaAlMaestro extends Behaviour
 {
        private int step=1;
        private int stat=0;
        public void action() {
        MessageTemplate mt
        =MessageTemplate.MatchPerformative(ACLMessage.CFP);
        switch (step)
        {
        case 1:
                if (msg!= null) {
                target = msg.getContent();
                System.out.println("target:"+target+ getAID().getName());
                step++;
                }
```

```
        Else
         {
        System.out.println("blokeo1"+ getAID().getName());
        block();
        }
        break;
case 2:
        msg= myAgent.receive(mt);
        if (msg != null)
        {
        X= msg.getContent();
        System.out.println("vector x:"+X+ getAID().getName());
        step++;
        }
        Else
         {
        System.out.println("blokeo2"+ getAID().getName());
        block();
        }
        break;
case 3:
        msg= myAgent.receive(mt);
        if (msg != null)
         {
        Y = msg.getContent();
        System.out.println("vector y:"+Y+ getAID().getName());
        step++;
        }
        else
        {
        System.out.println("blokeo3"+ getAID().getName());
        block();
        }
        break;
case 4:
        msg= myAgent.receive(mt);
        if (msg != null) {
        Npobladores=Integer.parseInt(msg.getContent());
        System.out.println("pobladores:"+Npobladores+ getAID().getName());
        step++;
        }
        else
        {
        System.out.println("blokeo4"+ getAID().getName());
        block();
```

```
        }
        break;
case 5:
        msg= myAgent.receive(mt);
        if (msg != null)
         {
        Nciudades=Integer.parseInt(msg.getContent());
        System.out.println("ciudades:"+Nciudades+ getAID().getName());
        step++;
        msg.createReply();
        mTsp= new int [Nciudades][2];// este es el problema entorno
        padre = new int[Nciudades];
        madre = new int[Nciudades];
        hijo = new int[Nciudades];
        hija = new int[Nciudades];
        Mejor= new int[Nciudades];
        mPoblacion= new int [Npobladores][Nciudades];
        GenNext= new int [Npobladores][Nciudades];
        }
        Else
         {
        block();
        }
        break;
case 6:
        Distance Distancia=new Distance(Npobladores,Nciudades);
        Fitnes ajuste=new Fitnes(Npobladores,Nciudades);
        FirstGeneration Poblacion= new
        FirstGeneration(Npobladores,Nciudades);
        Nearest cercano=new Nearest(Nciudades);
        for (int k=0; k<Nciudades;k++)
        {
        mTsp[k][0] = Integer.parseInt(X.split(",")[k].trim());
        mTsp[k][1] = Integer.parseInt(Y.split(",")[k].trim());
        Mejor[k]=Integer.parseInt(target.split(",")[k].trim());
        }
        Poblacion.Generation();//primera generacion
        mPoblacion=Poblacion.obtenerPoblado();
        for(int h=0;h<Npobladores;h++)
        {
        mPoblacion[h]=cercano.Cercano(mTsp, mPoblacion[h]);
        }
        Distancia.CalcularDistancia(mTsp, mPoblacion);
        distancias=Distancia.getDistance();
        ajuste.calculaFitnes(distancias);
```

```
            worse=ajuste.getWorse();
            step++;
            break;
    case 7:
            Distance Distancia1=new Distance(Npobladores,Nciudades);
            Fitnes ajuste1=new Fitnes(Npobladores,Nciudades);
            Nearest cercano1=new Nearest(Nciudades);
            CopyVector copiar= new CopyVector(Nciudades,Npobladores);
            CopyVector copia= new CopyVector(Nciudades,Npobladores);
            Roulett Ruleta=new Roulett(Npobladores,Nciudades);
            Random random = new Random();
            Selection seleccion=new Selection(Npobladores);
            Cross cruce=new Cross();
            Mutation mutacion= new Mutation();
            mPoblacion[worse]=copiar.CopiarVector(Mejor);
            Distancia1.CalcularDistancia(mTsp, mPoblacion);
            distancias=Distancia1.getDistance();
            ajuste1.calculaFitnes(distancias);
            Best=ajuste1.getBest();
            worse=ajuste1.getWorse();
            fitnes=ajuste1.getFit();
            Ruleta.Ruleta(fitnes);
            ruleta=Ruleta.getRoul();
            Mejor=copia.CopiarVector(mPoblacion[Best]);
            GenNext[0]=copiar.CopiarVector(Mejor);
            do
            {
            pcruz=Math.abs(random.nextInt()%(101));
            }
            while (pcruz<90);
            pmut=Math.abs(random.nextInt()%(15));
            int n=(Npobladores*pcruz)/100;
            for(int k=1;k<n;k++ )//nueva generacion
            {
            padre1=seleccion.seleccion(ruleta);
            padre2=seleccion.seleccion(ruleta);
            padre=copiar.CopiarVector(mPoblacion[padre1]);
            madre=copiar.CopiarVector(mPoblacion[padre2]);
            cruce.Cruce(padre, madre);
            hijo=cruce.getHijo(0);
            hija=cruce.getHijo(1);
            Distancia1.CalculaVec(mTsp,hijo);
            dish1=Distancia1.getVDistance();
            Distancia1.CalculaVec(mTsp,hija);
            dish2=Distancia1.getVDistance();
```

```
prob = Math.abs(random.nextInt()%(101));
if(dish1<=dish2)
{
if(dish1<distancias[padre1]&&dish1<distancias[padre2])
        {
        if (prob<=pmut)
        {
        hija= mutacion.mutacion(hija, 1);
        muta++;
        }
GenNext[k]=copiar.CopiarVector(hija);
cruz++;
 }
else
{
if(distancias[padre1]<=distancias[padre2])
{
if (prob<=pmut)
 {
padre= mutacion.mutacion(padre, 1);
muta++;
}
GenNext[k]=copiar.CopiarVector(padre);
}
else
{
if (prob<=pmut)
{
madre= mutacion.mutacion(madre, 1);
muta++;
}
GenNext[k]=copiar.CopiarVector(madre);
}
}
}
else
{
if(dish2<distancias[padre1]&&dish2<distancias[padre2])

{

if (prob<=pmut)
{
hijo= mutacion.mutacion(hijo, 1);
muta++;
```

```
}
GenNext[k]=copiar.CopiarVector(hijo);
cruz++;
}
            else
{
if(distancias[padre1]<=distancias[padre2])
{
if (prob<=pmut)
{
padre= mutacion.mutacion(padre,1);
muta++;
}
GenNext[k]=copiar.CopiarVector(padre);
}
else
{
if (prob<=pmut)
{
madre= mutacion.mutacion(madre, 1);
muta++;

}
GenNext[k]=copiar.CopiarVector(madre);

 }
 }
}
}
for(int p=n;p<Npobladores;p++)
{
GenNext[p]=copiar.CopiarVector(Mejor);
GenNext[p]=mutacion.mutacion(GenNext[p],Npobladores);
GenNext[p]=cercano1.Cercano(mTsp, GenNext[p]);
}
mPoblacion=copiar.CopiarMatriz(GenNext);
Distancia1.CalcularDistancia(mTsp, mPoblacion);
distancias=Distancia1.getDistance();
ajuste1.calculaFitnes(distancias);
fitnes=ajuste1.getFit();
Best=ajuste1.getBest();
worse=ajuste1.getWorse();
Ruleta.Ruleta(fitnes);
ruleta=Ruleta.getRoul();
mejor=copia.CopiarVector(mPoblacion[Best]);
```

```
//mejor individuo d eesta generacion
        target="";
        for(int j=0;j<Nciudades;j++)
        {
        target=target+Integer.toString(Mejor[j]);
         if(j<Nciudades-1)
        {
        target=target+',';
        }
        }
        reply=msg.createReply();
        reply.setContent(target);
        myAgent.send(reply);
        step++;
        System.out.println("envio target"+ getAID().getName());
        break;

case 8://recibe un 9 para continuar o un 10 para salir
        msg= myAgent.receive(mt);
        if (msg != null)
        {
        step=Integer.parseInt(msg.getContent());
        }
        Else
         {
        block();
        }
        break;

case 9:
        MessageTemplate.MatchPerformative(ACLMessage.CFP);
        msg= myAgent.receive(mt);
        if (msg != null)
         {
        target = msg.getContent();
        System.out.println(target);
        for (int k=0; k<Nciudades;k++)
        {
        Mejor[k]=Integer.parseInt(target.split(",")[k].trim());
        step=7;
        }
        else {
        block();
        }
        break;
```

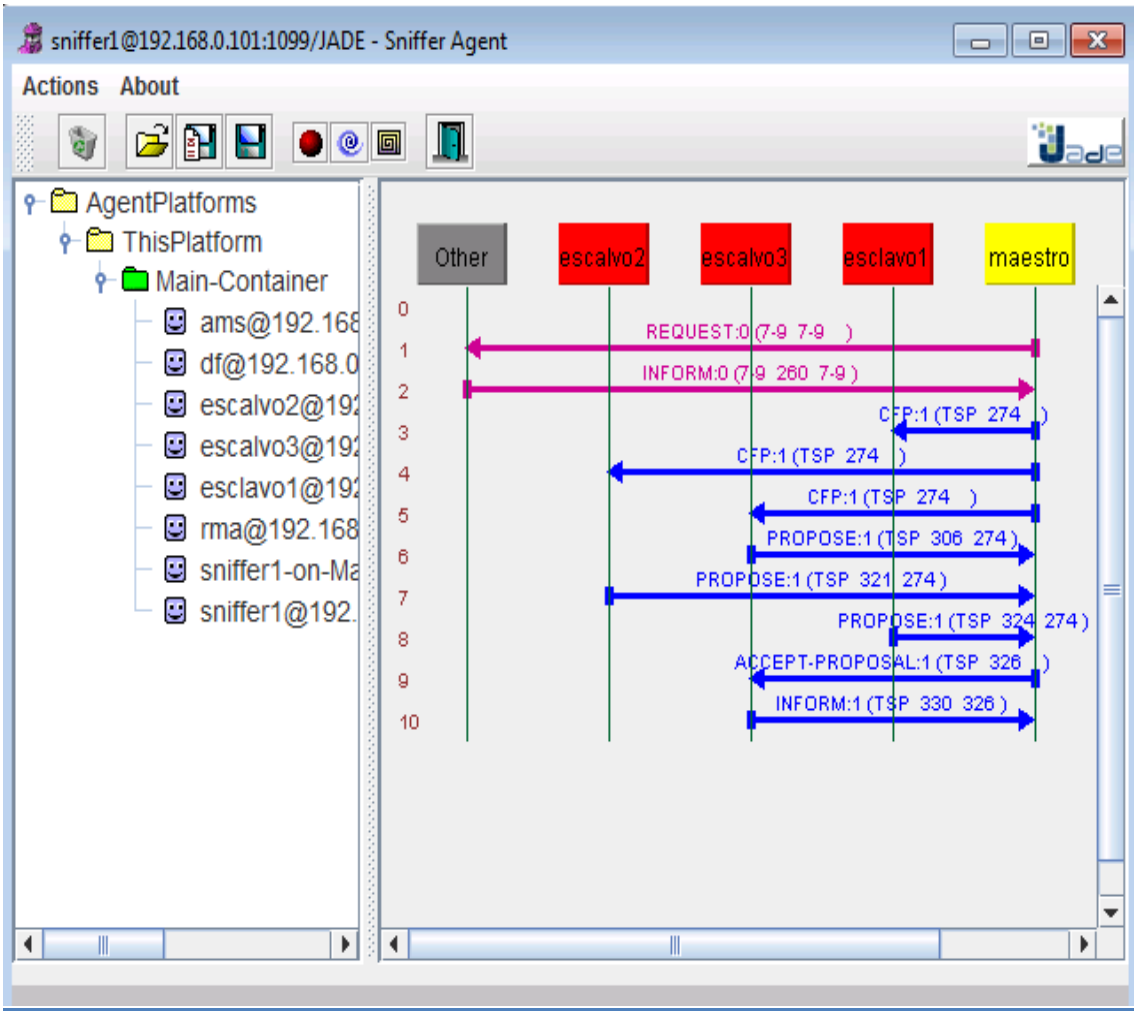
```
        case 10:
                break;
                }
        }
        @Override
public boolean done()
{
return step == 10;
}

public int onEnd()
{
System.out.println("termina agente S"+ getAID().getName());
myAgent.doDelete();
return super.onEnd();
}
}
}
```

ANEXO D.

Pruebas de comunocacion entre agentes sobre la plataforma JADE.

Sobre la plataforma JADE se realizaro pruebas de comunicación entre los agentes obteniéndose el diagrama de interacciones mostrado en la **figura 46** lo cual se hace con un agente herramienta implementado sobre la plataforma llamado agente *SNIFFER.*

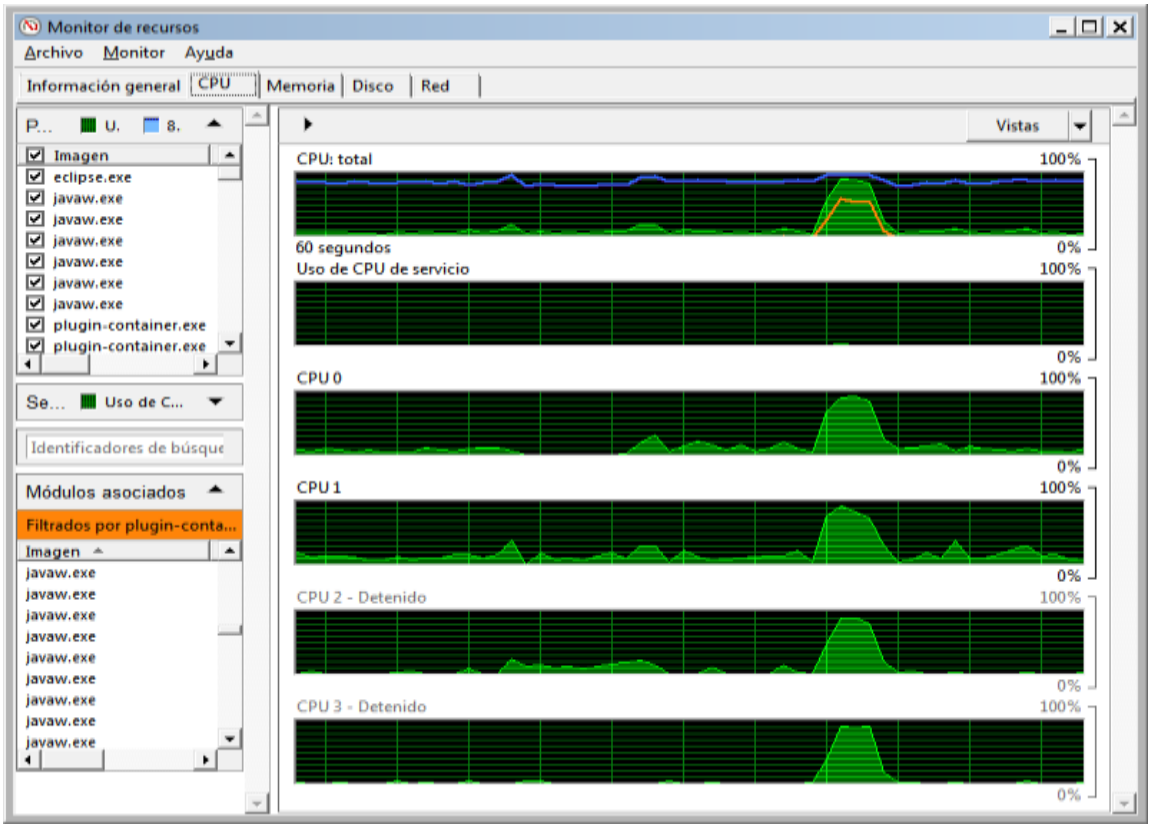
Figura 38 Interacciones SMA sobre JADE.



ANEXO E.

Uso de la CPU al ejecutar el sistema multiagente software (**figura 41**) con el problema oliv30, con 60 individuos y 300 generaciones y cuatro agentes esclavo.

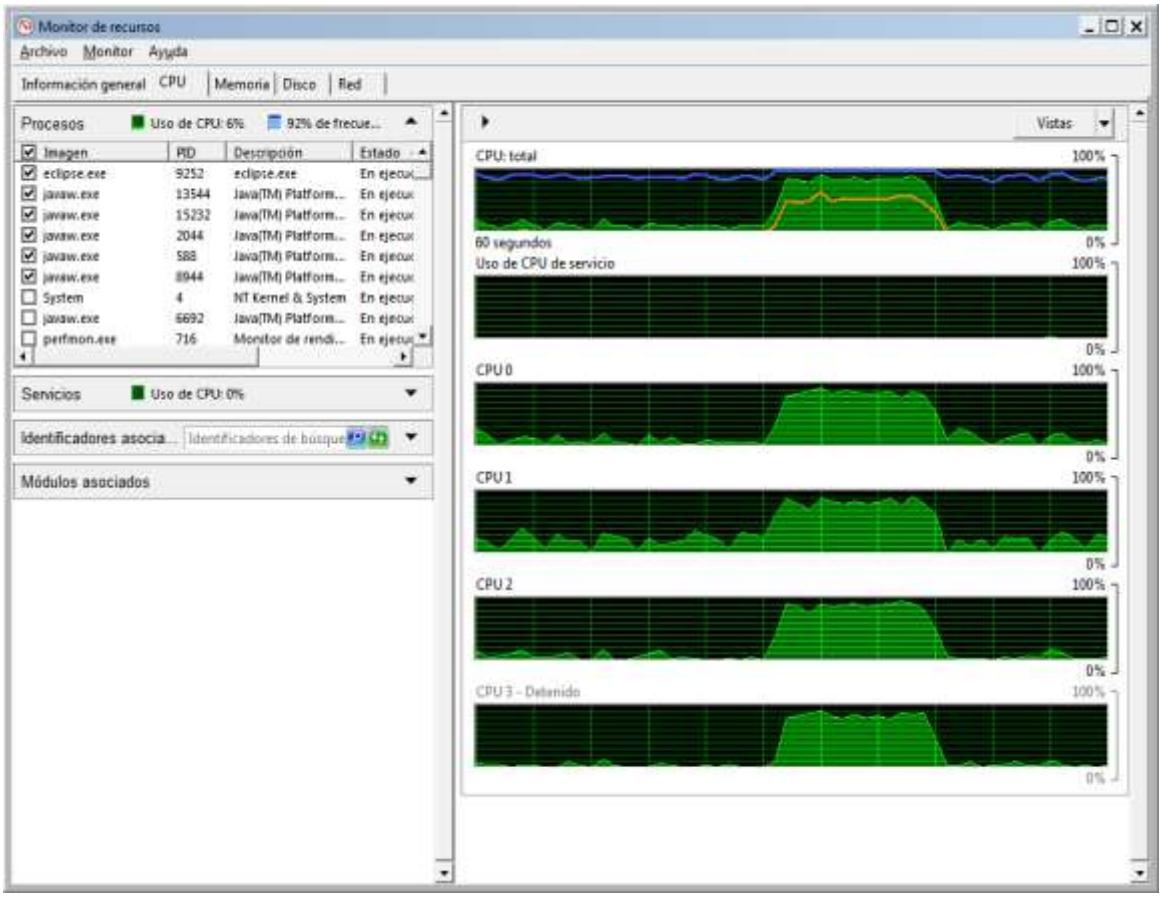
Figura 39 Uso de la CPU durante ejecucion del SMA software en JADE.



Durante los 3.5 segundos que dura la ejecucion del programa para estas condiciones, el procesador es exigido al máximo en sus 4 núcleos.

Del mismo modo al ejecutar el SMA con el problema eil51 bajo las mismas condiciones se observa que el los recursos de la máquina durante los 12 segundos de ejecución es llevado también al máximo de rendimiento.

Figura 40 Uso de la CPU durante ejecucion del SMA software en JADE.



ANEXO F.

Descripción VHDL del agente maestro

```
LIBRARY  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

DELCALRACIÓN DE LA ENTIDAD DEL AGENTE MAESTRO

```
entity master is
port(
-- entradas
distance: out std_logic_vector(13 downto 0);
as1: in std_logic_vector(13 downto 0);--puerto de recepcion agente1
as2: in std_logic_vector(13 downto 0);--puerto de recepcion agente2
as3: in std_logic_vector(13 downto 0);--puerto de recepcion agente3
as4: in std_logic_vector(13 downto 0);--puerto de recepcion agente4
qa1:in std_logic;
qa2:in std_logic;
qa3:in std_logic;
qa4:in std_logic;
mind: in std_logic_vector(6 downto 0);
mciu: in std_logic_vector(6 downto 0);
mgen: in std_logic_vector(13 downto 0);
clk: in std_logic; -- reloj del sistema
start: in std_logic;-- orden de inicio
reset: in std_logic;-- orden de inicio
ready: in std_logic_vector(6 downto 0);-- lectura coordenadas datos
readx: in std_logic_vector(6 downto 0);-- lectura coordenadas datos
ndin1:in std_logic;--proximo dato de entrada
ndin2:in std_logic;--proximo dato de entrada
```

```
ndin3:in std_logic;--proximo dato de entrada
ndin4:in std_logic;--proximo dato de entrada
--salidas
dirdat: out std_logic_vector (6 downto 0);--puerto de lectura de datos del problema
coma: out std_logic_vector(13 downto 0);--puerto de envio de datos a los agentes
rqa1:out std_logic;
final:out std_logic;
ndou1:out std_logic;--proximo dato de salida
 xp11:out std_logic_vector (6 downto 0);
yp11: out std_logic_vector(6 downto 0);
xp21: out std_logic_vector (6 downto 0);
yp21: out std_logic_vector(6 downto 0);
xp12:out std_logic_vector (6 downto 0);
yp12: out std_logic_vector(6 downto 0);
xp22: out std_logic_vector (6 downto 0);
yp22: out std_logic_vector(6 downto 0);
xp13: out std_logic_vector (6 downto 0);
yp13: out std_logic_vector(6 downto 0);
xp23: out std_logic_vector (6 downto 0);
yp23: out std_logic_vector(6 downto 0);
xp14: out std_logic_vector (6 downto 0);
yp14: out std_logic_vector(6 downto 0);
xp24: out std_logic_vector (6 downto 0);
yp24: out std_logic_vector(6 downto 0);
maxciudades:out std_logic_vector(6 downto 0);
maxindividuos:out std_logic_vector(6 downto 0)
);
end entity master;
```

DECLARACIÓN DE LA ARQUITECTURA DEL AGENTE MAESTRO

{

Declaración de componentes del agente maestro, unidades funcionales.

{

Component componente 1

Component componete 2

.

.

Component componete n

}

architecture maestro of master is

{

Declaración de señales para interconectar los componentes.

Signal señal 1

Signal señal 2

Signal señal 3

.

.

Signal señal n

}

Begin

Concatenacion de los componentes mediante las señales requeridas.

CREENCIAS E INTENCIONES del agente compuestos por los módulos constitutivos m1, m2, m3, m4 y el arregla de memoria de problema loadproblem

{

m1: componenteM port map(done1,best1,dto1,dira1,dirb1,start1,ndin1,as1(13 downto 7),as1(6 downto 0),

```
x1m1,x2m1,y1m1,y2m1,maxc,maxi,clk,envia1,recibe1,findbest1,reset,nresp);

m2: componenteM port map(done2,best2,dto2,dira2,dirb2,start2,ndin2,
                as2(13 downto 7),as2(6 downto
0),x1m2,x2m2,y1m2,y2m2,maxc,maxi,clk,envia1,recibe1,findbest1,reset,nresp);

m3: componenteM port map(done3,best3,dto3,dira3,dirb3,start3,ndin3,
as3(13 downto 7),as3(6 downto
0),x1m3,x2m3,y1m3,y2m3,maxc,maxi,clk,envia1,recibe1,findbest1,reset,nresp);

m4:componenteM port map(done4,best4,dto4,dira4,dirb4,start4,ndin4,
                as4(13 downto 7),as4(6 downto
0),x1m4,x2m4,y1m4,y2m4,maxc,maxi,clk,envia1,recibe1,findbest1,reset,nresp);

load: loadproblem port
map(clk=>clk,start=>start,reset=>reset,maxi=>mind,maxc=>mciu,maxg=>mgen,
diram1=>mx1(13 downto 7),diram2=>mx2(13 downto 7),diram3=>mx3(13 downto
7),diram4=>mx4(13 downto 7)
,dirbp1=>mx1(6 downto 0),dirbp2=>mx2(6 downto 0),dirbp3=>mx3(6 downto
0),dirbp4=>mx4(6 downto 0),
datax=>readx,datay=>ready,maxio=>maxi,maxco=>maxc,maxgo=>maxg,xp11=>x
1m1,yp11=>y1m1,xp21=>x2m1,yp21=>y2m1
,xp12=>x1m2,yp12=>y1m2,xp22=>x2m2,yp22=>y2m2,xp13=>x1m3,yp13=>y1m3,
xp23=>x2m3,yp23=>y2m3,xp14=>x1m4,yp14=>y1m4,xp24=>x2m4,
yp24=>y2m4,dirmem=>dirdata,done=>finl);
}
```

INTERACCIONES del agente que detemina los datos de entrada y de salida multiplexando a la salida la major poblacion. También se encarga de la detección de agentes esclavo conectados a el y que poresten el servicio requerido (TSP).

{

```
muxo: muxndo port map (dto1,dto2,dto3,dto4,sel,ndou1);
dec: decocom Port map(da1(13 downto 7)=>dira1,da1(6 downto 0)=>dirb1,da2(13
downto 7)=>dira2,da2(6 downto 0)=>dirb2,da3(13 downto 7)=>dira3,
da3(6 downto 0)=>dirb3,da4(13 downto 7)=>dira4,da4(6 downto
0)=>dirb4,a1=>p1,b2=>p2,c3=>p3,d4=>p4,best=>mejor,sel=>sel,best1=>distancia
);
muxa:remux port map(sel1=>qa1,sel2=>qa2,sel3=>qa3,sel4=>qa4,dat1=>best1,
dat2=>best2 ,dat3=>best3,
dat4=>best4,dato1=>p1,dato2=>p2,dato3=>p3,dato4=>p4);
generaciones:contador14 port map(clr,ldc,cle,clk,datl,cuenta);
fgenera: compara14 port map(cuenta,maxg,fing);
dirdat<=dirdata;
coma<=mejor;
ldc<='0';
datl<="00000000000000";
anddone<=(done1 and done2 and done3 and done4);
respsla<=(qa1 or qa2 or qa3 or qa4);
 final<=fin;
 rqa1<=rqs;
 distance<= distancia;
 nxdi<=ndin1 and ndin2 and ndin3 and ndin4;
 nresp<=nxdi;
xp11<=x1m1;
xp21<=x2m1;
yp11<=y1m1;
yp21<=y2m1;
xp12<=x1m2;
xp22<=x2m2;
yp12<=y1m2;
```

```
yp22<=y2m2;
xp13<=x1m3;
xp23<=x2m3;
yp13<=y1m3;
yp23<=y2m3;
xp14<=x1m4;
xp24<=x2m4;
yp14<=y1m4;
yp24<=y2m4;
maxciudades<=maxc;
maxindividuos<=maxi;
}
```

FSM que representa los DESEOS del agente maestro mediante estados mentales concretos cuyos comportameintos están dentro de cada modulo "m".

```
{
fsm: fsmmaster port
map(clk,anddone,finl,fing,reset,respsla,findbest1,recibe1,envia1,
start1,start2,start3,start4,rqs,clr,cle,fin,sela);
mux1:muxre port map(data0x=>as1,data1x(13 downto 7)=>dira1,data1x(6 downto
0)=>dirb1,sel=>sela,result=>mx1);
mux2:muxre port map(data0x=>as2,data1x(13 downto 7)=>dira2,data1x(6 downto
0)=>dirb2,sel=>sela,result=>mx2);
mux3:muxre port map(data0x=>as3,data1x(13 downto 7)=>dira3,data1x(6 downto
0)=>dirb3,sel=>sela,result=>mx3);
mux4:muxre port map(data0x=>as4,data1x(13 downto 7)=>dira4,data1x(6 downto
0)=>dirb4,sel=>sela,result=>mx4);
}
end architecture;
```

ANEXO G.

Algoritmo de green implementado en hardware como generador de numeros pseudo aleatorios.

El algoritmo de Green es un generador de números pseudo aleatorios aditivo, en la que su salida emplea 16 semillas anteriores siendo de la forma:

$$x_{n+1} = (x_n + x_{n-k}) \, mod \, c$$

Donde ***k*>15**

Las 16 semillas anteriores son generadas por medio de un algoritmo congruencial lineal de la forma:

$$x_{n_{+1}} = a * x_n + b \, mod \, c$$

De esta forma la máquina de estados se encarga de encontrar las primeras 16 semillas con el algoritmo congruencial lineal y a partir de ese momento la secuencia generada será la de Green, las constantes tomadas para el algoritmo congruencial lineal son a = 5, b = 1 y c = 2^{15}, la semilla es tomada de un contador cuando el usuario hace la petición de solucionar un problema, de esta manera el agente es autónomo en la selección del estos parámetros.

Las pruebas realizadas a este módulo en **Modelsim,** arroja resultados deseables de un generador pseudo aleatorio (**figura 38**). La simulación se realizó con 2000 muestras en el rango 0 -100.

En la **figura 39** se tiene la máquia de estados implementada en el RNG de *Green*.

Figura 41 Simulación algoritmo de Green hardware.

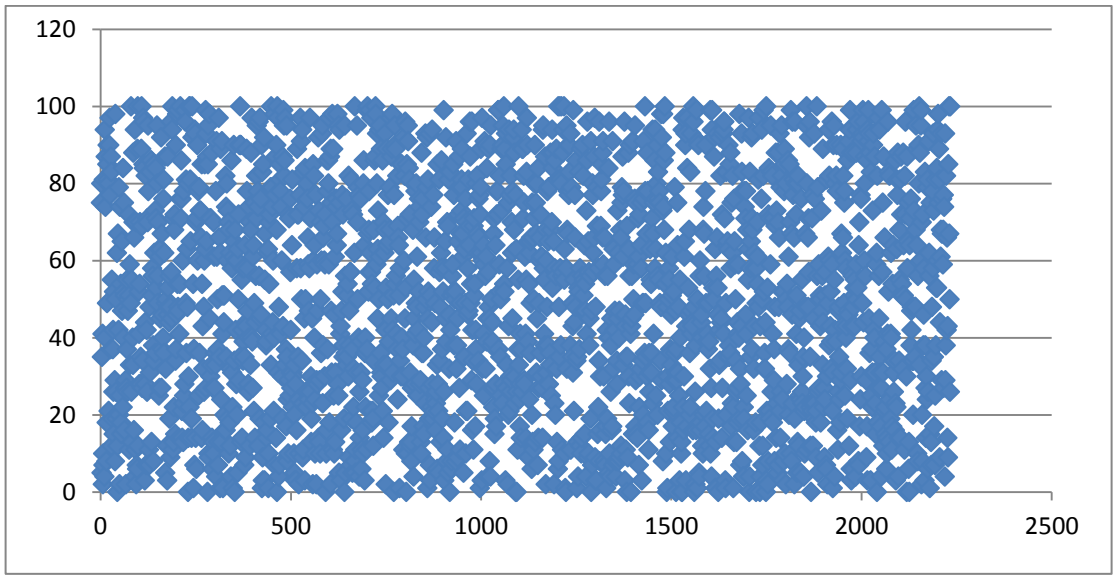
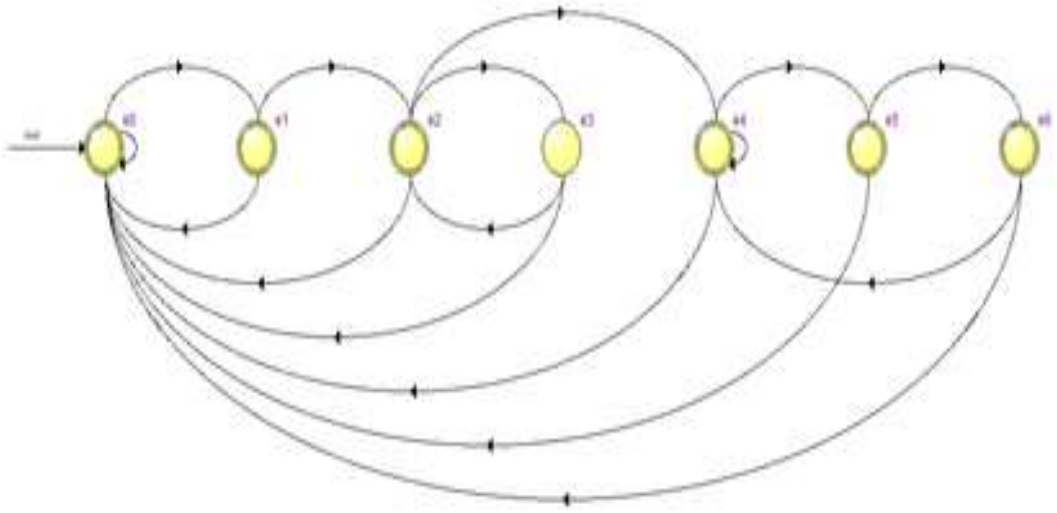
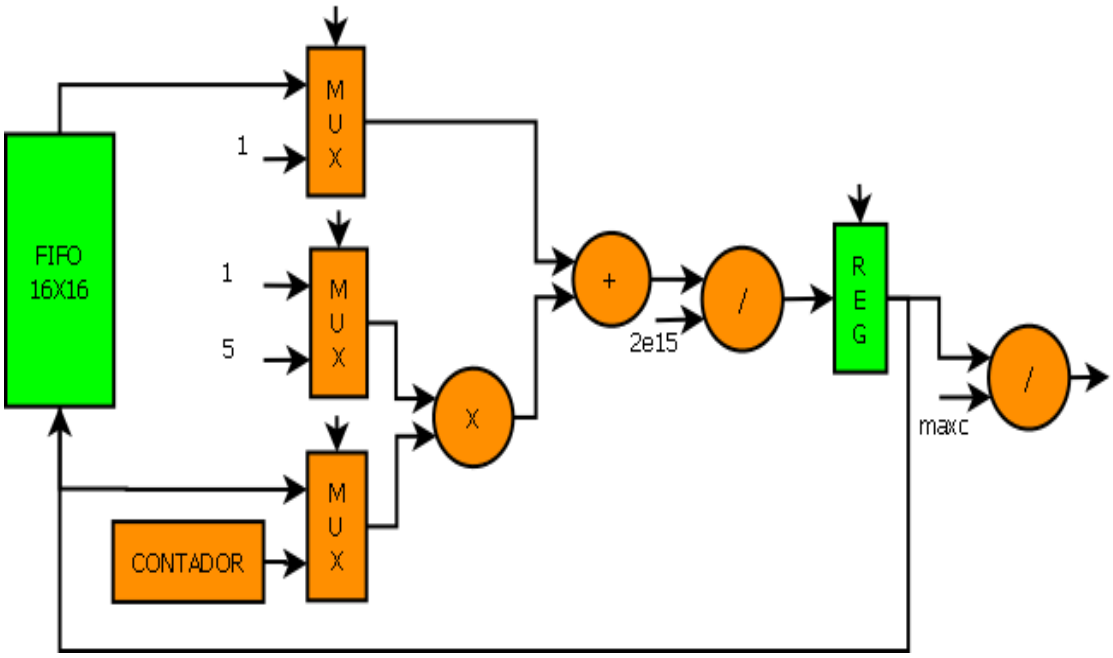


Figura 42 Máquina de estados algoritmo Green.



Circuito implementado para el RNG de *Green (***figura 40***)*, teniendo como algoritmo de inicialización de las primeras 16 semillas un algoritmo congruencial lineal.

Figura 43 Diagrama esquemático implementación algoritmo de Green.



ANEXO H.

Descripcion en VHDL del agente esclavo

```
LIBRARY  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

DECLARACIÓN DE LA ENTIDAD DEL AGENTE ESCLAVO

```
entity slave is
port(
-- entradas
start:in std_logic;
dadb: in std_logic_vector(13 downto 0);--recepcion de datos de memoria ram de
master
qa1:in std_logic;--peticion de master
mind: in std_logic_vector(6 downto 0);--numero de individuos
mciu: in std_logic_vector(6 downto 0);--numero de ciudades
clk: in std_logic; -- reloj del sistema
ndin1:in std_logic;--proximo dato de entrada
--salidas
rqa1:out std_logic;-- respuesta al maestro
ndou1:out std_logic;--proximo dato de salida
 x1: in std_logic_vector(6 downto 0);--coordenadas
 x2: in std_logic_vector(6 downto 0);--coordenadas
 y1: in std_logic_vector(6 downto 0);--coordenadas
 y2: in std_logic_vector(6 downto 0);--coordenadas
 salida:out std_logic_vector(13 downto 0)
);
end entity slave;
```

DECLARACIÓN DE LA ARQUITECTURA DEL AGENTE ESCLAVO

architecture esclavo of slave is

{

Declaración de componentes del agente, unidades funcionales

Component componente 1

Component componente 2

.

.

Component componente n

}

{

Declaración de señales para interconeccion de componentes del agente esclavo.

Signal señal 1

Signal señal 2

.

.

Signal señal n

}

Begin

{

CREENCIAS E INTENCIONES del agente esclavo, concatenacion de los componentes por medio de las señales y compuestos por el módulos constitutivo componentes y el arregla de memoria de problema loadproblem

{

componente: componenteS port map(done=>ok,ndataout=>ndo,dataA=>bestd(13 downto 7),dataB=>bestd(6 downto 0),start=>ini,ndatain=>ndin1,pobin1=>dadb(13 downto 7),pobin2=>dadb(6 downto)
,x1=>x1,x2=>x2,Y1=>y1,y2=>y2,maxc=>mciu,maxi=>mind,clk=>clk,envia=>envia,recibe=>recibe,findbest=>findbest,reset=>reset,stra1=>start);

}

INTERACCIONES del agente que detemina los datos de entrada y de salida. También se encarga de registrar ante un maestro su estado de aceptar o rechazar un problema (TSP).

{

salida<=bestd;

rqa1<=rqa;

ndou1<=ndo;

}

FSM que representa los DESEOS del agente esclavo mediante estados mentales concretos cuyos comportameintos están dentro del módulo "componeteS".

{

fsm: fsmslave port map (start,clk,ok,qa1,findbest,recibe,envia,ini,rqa,reset);

}

end architecture;

ANEXO I.

Resultados de la síntesis en QUARTUS II del sistema multiagente, agente maestro y agente esclavo (**figura 43, 44, 45**).

Figura 44 Resultado síntesis del SMA.

Figura 45 Resultado síntesis de agente esclavo.

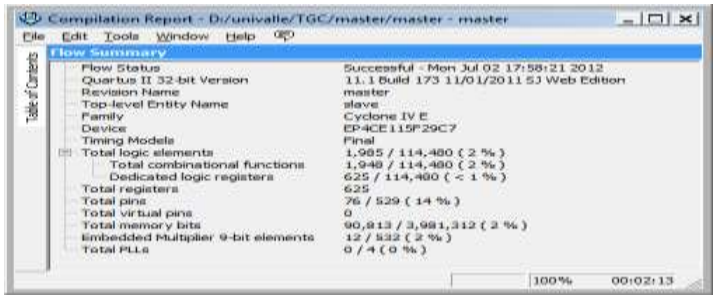
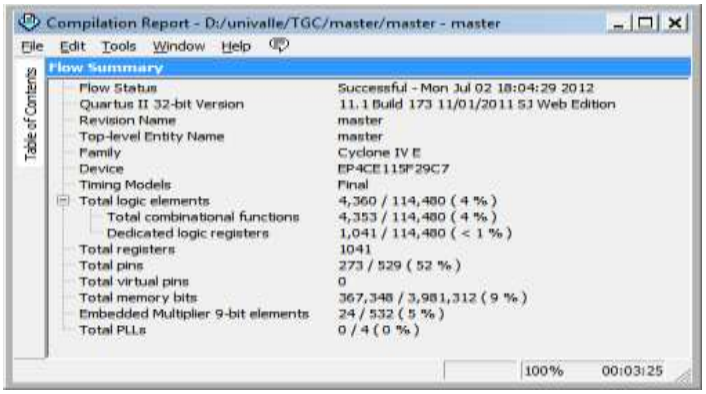


Figura 46 Resultado síntesis del agente maestro.



ANEXO J.

Herramienta para graficar la respuesta del SMA Hardware.

```
package geneticAl;
//import java.util.*;
import java.io.*;
import javax.swing.JFrame;
public class HW2 {
public static String target;
static int[][] mTsp, mPoblacion,Sol;//,poblado;
static int[] vector1S;
static int Nciudades;
static int Npobladores;
static int pmut;
static int alea;
static int gen,k1,k2,k3;
  public static void main(String[] args)throws IOException
        {
        ArchiveLoader Lectura = new ArchiveLoader();
        target="\\Users\\paula\\workspace\\geneticAL\\src\\geneticAl\\eil4.tsp";
        Lectura.LeerArchivo("C:"+target);
        mTsp = Lectura.getCiudades();
        Nciudades=Lectura.getNCiudades();
        System.out.println("numero de ciudades "+Nciudades);
        BufferedReader lectura = new BufferedReader(new
        InputStreamReader(System.in));
        String datos;
        System.out.println("Ingrese los datos: ");
        datos = lectura.readLine();
        int[] vectorS = new int[Nciudades];
        for (int k=0; k<Nciudades;k++)
        {
```

```
vectorS[k] = Integer.parseInt(datos.split(",")[k].trim());
}
Distancia.CalculaVec(mTsp, vectorS);
alea=Distancia.getVDistance();
Sol= new int[Nciudades][2];

for(int k=0;k<Nciudades;k++)
{
Sol[k][0]=mTsp[vectorS[k]-1][0];
Sol[k][1]=mTsp[vectorS[k]-1][1];
}
JFrame ventana = new JFrame();
Lienzo lienzo = new Lienzo(Sol,Nciudades);
ventana.add(lienzo);
ventana.setSize(800,650);
ventana.setTitle("SAM HARDWARE"+" Distancia: " +alea);
ventana.setVisible(true);
}
}
```